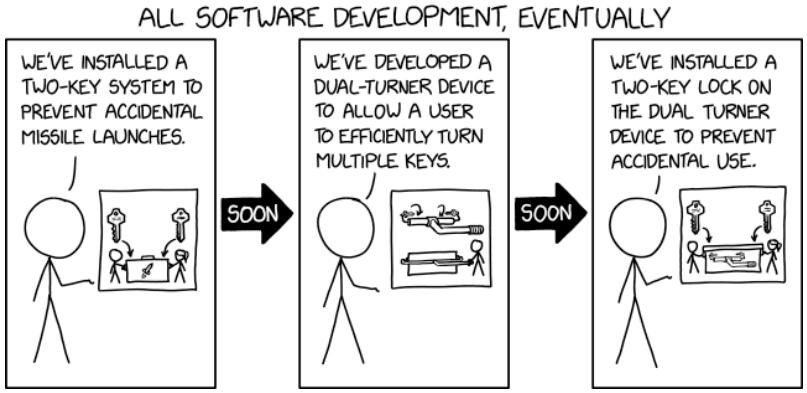
Primer Certamen
Algoritmos y Complejidad

15 de octubre de 2022



http://www.xkcd.com/2677

La entrega es en páginas separadas por pregunta, cada una debidamente identificada (nombre, rol, certamen y ramo). Si no responde, entregue una hoja en blanco.

1. Suponiendo $e \to 0$ (nos interesan valores pequeños de e) simplifique hasta términos $O(e^2)$ la siguiente expresión:

$$\frac{1+ae+be^2+O(e^3)}{1+ae+ce^2}\sqrt{1-2ae}$$

(15 puntos)

2. Las siguientes tres funciones tienen $\sqrt[3]{a}$ como punto fijo:

   a) $\frac{1}{3}\left(2x+\frac{a}{x^2}\right)$ b) $\frac{x^3}{a}+x-1$ c) $\sqrt{\frac{a}{x}}$

   ¿Cuál de las tres converge más rápido cerca del punto fijo si $a>0$?

(25 puntos)

3. Se propone una fórmula de cuadratura:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx A_{-1}f(-1)+A_0 f(0)+A_1 f(1)+Bf'(0)$$

   ¿Para qué grado máximo de polinomio puede ser exacta? ¿Cómo determinaría los coeficientes A_{-1}, A_0, A_1, B?

(15 puntos)

4. En clase mencionamos *Best First Search*. Describa el algoritmo del caso siguiendo los ejemplos del reverso. Se da el grafo $G=(V,E)$ y un vértice inicial s, con $N_G(v)$ los vecinos del vértice v, suponiendo funciones $g(v)$ que indica si el vértice v es una meta, $f(v)$ que da el valor de la meta v (buscamos minimizar $f(v)$) y una heurística $h(v)$ que indica lo deseable que es continuar desde v (menor valor de h es más deseable). Suponga dadas estructuras básicas como conjuntos (use las operaciones comunes), pilas (`Stack`, con operaciones `push(S, e)`, donde `e` es el elemento, y `pop(S)` elimina y retorna el tope), colas (`Queue`, operaciones `enqueue(Q, e)`, con `e` el elemento, y `dequeue(Q)` elimina y retorna el primero), y colas de prioridad (`PriorityQueue`, con operaciones `enqueue(Q, e, p)`, con `e` el elemento y `p` su prioridad, y `deletemin(Q)` elimina y retorna el de menor prioridad).

(30 puntos)

5. Hay un conjunto de tareas a ser ejecutadas por una única máquina capaz de efectuar una tarea a la vez, todas de duración 1. La tarea i tiene plazo fatal t_i, y da una ganancia de v_i si se completa a tiempo (no aporta nada si se completa después). Hallar la programación de tareas que da la máxima ganancia puede hacerse mediante un algoritmo voraz. Esboce un algoritmo para resolver este problema, y demuestre que da un óptimo.

(35 puntos)

HvB/LaTeX2ε

Algoritmos ejemplo

Algoritmo 1: Búsqueda en profundidad – versión recursiva

```
C ← ∅
procedure DFS(G, s)
    C ← C ∪ {s}
    process(s)
    foreach v ∈ N_G(s) do
        if v ∉ C then
            DFS(G, v)
        end
    end
end
```

Algoritmo 2: Búsqueda en profundidad – versión iterativa

```
procedure DFS(G, s)
    S ← Stack; push(S, s); C ← ∅
    while ¬empty(S) do
        s ← pop(S)
        if s ∉ C then
            C ← C ∪ {s}
            process(s)
            foreach v ∈ N_G(s) do
                if v ∉ C then
                    push(S, v)
                end
            end
        end
    end
end
```

Algoritmo 3: Búsqueda a lo ancho

```
procedure BFS(G, s)
    Q ← Queue; enqueue(Q, s); C ← ∅
    while ¬empty(Q) do
        s ← dequeue(Q)
        if s ∉ C then
            C ← C ∪ {s}
            process(s)
            foreach v ∈ N_G(s) do
                if v ∉ C then
                    enqueue(Q, v)
                end
            end
        end
    end
end
```

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

15 de octubre de 2022

1. Son manipulaciones rutinarias:

$$\begin{aligned}
\frac{1+ae+be^2+O(e^3)}{1+ae+ce^2}\sqrt{1-2ae} &= \frac{1+ae+O(e^2)}{1+ae+O(e^2)}\left(1-\frac{1}{2}\cdot 2ae+O(e^2)\right) \\
&= (1+ae+O(e^2))(1-ae+O(e^2))(1-ae+O(e^2)) \\
&= (1-a^2e^2+O(e^2))(1-ae+O(e^2)) \\
&= (1+O(e^2))/1-ae+O(e^2)) \\
&= 1-ae+O(e^2)
\end{aligned}$$

Puntajes

Total		15
Expandir $\sqrt{\cdot}$	4	
Expandir $1/\cdot$	4	
Operar el producto	2	
Simplificar el resultado	5	

2. Primero, sabemos que la iteración $x_{n+1} = g(x_n)$ converge a x^* cerca de x^* si $g'(x)$ es continua en x^* y $|g'(x^*)| < 1$. Sabemos además que la convergencia cerca de x^* es más rápida cuanto menor es $|g'(x^*)|$. Si $g'(x^*) = 0$, el orden de convergencia es más que lineal. Analicemos cada una de las funciones por turno en $x^* = \sqrt[3]{a}$:

   a)
   $$g_a(x) = \frac{1}{3}\left(2x + \frac{a}{2x^2}\right)$$
   $$g_a'(x) = \left(2 + \frac{-2a}{2x^3}\right)$$
   $$g_a(x^*) = 0$$

   Esto indica convergencia más que lineal.

   En realidad, este es el método de Newton aplicado a $x^3 - a$, de convergencia cuadrática.

   b)
   $$g_b(x) = \frac{x^3}{a} + x - 1$$
   $$g_b'(x) = \frac{3x^2}{a} + 1$$

   Es claro que $g_b'(x) \geq 1$ para todo $a > 0$ nunca converge.

   c)
   $$g_c(x) = \sqrt{\frac{a}{x}}$$
   $$g_c'(x) = \frac{1}{2}\left(\frac{x}{a}\right)^{1/2} \cdot \left(-\frac{a}{x^2}\right)$$
   $$= -\frac{1}{2}\frac{x^{1/2}a}{a^{1/2}x^2}$$
   $$= -\frac{1}{2}\frac{a^{1/2}}{x^{3/2}}$$
   $$g_c'(x^*) = -\frac{1}{2}$$

   Esto converge linealmente para todo a.

Puntajes

Total		25
a)	Más que lineal	9
b)	No converge	8
c)	Convergencia lineal	8

3. La fórmula tiene 4 coeficientes a determinar, el grado máximo de polinomio al que puede aspirar es 3. Una forma de hallar los coeficientes es plantear un sistema de ecuaciones:

$$\begin{aligned} A_{-1} + A_0 + A_1 &= \int_{-1}^{1} \mathrm{d}x = 2 \\ -A_{-1} + A_1 + B &= \int_{-1}^{1} x\,\mathrm{d}x = 0 \\ A_{-1} + A_1 &= \int_{-1}^{1} x^2\,\mathrm{d}x = \tfrac{2}{3} \\ -A_{-1} + A_1 &= \int_{-1}^{1} x^3\,\mathrm{d}x = 0 \end{aligned}$$

Lamentablemente se ve del sistema de ecuaciones que $B = 0$, el resultado es simplemente la regla de Simpson.

Puntajes

Total	30
Son 4,000 00 coeficientes, cúbica	10
Plantear sistema	20
Resulta $B = 0$, es la regla de Simpson	5

4. El algoritmo es una variación de búsqueda en profundidad, en vez de ir ciegamente por el primer vértice los considera en orden de h creciente. Usamos la variable m para registrar el mínimo hallado hasta el momento, t es el vértice meta que da el mínimo hasta este punto. Procesar el vértice es revisar si es meta, si lo es actualizar la mejor hallada. Vea el algoritmo 1.

Algoritmo 1: Búsqueda mejor primero

```
function BFS(G, s)
    Q ← PriorityQueue
    C ← ∅; m ← ∞
    enqueue(Q, s, h(s))
    while ¬empty(Q) do
        s ← deletemin(Q)
        if s ∉ C then
            C ← C ∪ {s}
            if g(s) ∧ f(s) < m then
                m ← f(s)
                t ← s
            end
            foreach v ∈ N_G(s) do
                if v ∉ C then
                    enqueue(Q, v, h(v))
                end
            end
        end
    end
    return t
end
```

Puntajes

Total	30
Explicación	5
Manejo de PriorityQueue	15
Qué es procesar el vértice	10

5. Conviene considerar las tareas en orden de ganancia decreciente, ubicando la tarea elegida en la última ranura de tiempo disponible (si la hay) antes de su plazo fatal. Si hay empate de ganancias, elija primero la tarea con plazo fatal más tardío.

   Demostramos que da un óptimo demostrando las tres propiedades:

   Greedy choice: Por contradicción. Si la tarea de máxima ganancia aún sin programar no tiene espacio en que puede ser asignada, no hay nada que demostrar. Sea k la tarea elegida a ser ubicada en $t \leq t_k$. Si no hay tareas posteriores en orden de ganancia decreciente que puedan llenar ese espacio, el óptimo incluye esa asignación. Si hay otra tarea que puede llenar el espacio t, tendrá $v \leq v_k$, podemos intercambiarlas y la ganancia total no disminuye.

   Inductive substructure: Al ubicar una tarea, obtenemos el problema restante eliminando las tareas que ya no pueden programarse antes de sus plazos. No hay restricciones externas en el resultado.

   Obtimal substructure: La solución óptima al subproblema que se obtiene al programar la elección voraz puede combinarse con la elección voraz, por las anteriores no hay interferencias y no se puede obtener nada mejor.

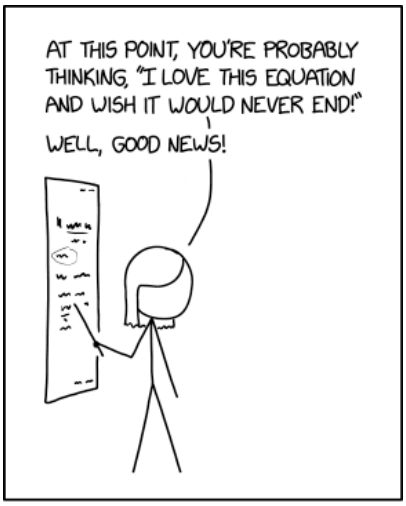
   Como cumple las tres propiedades, el algoritmo voraz da un óptimo.

Puntajes

Total	30
Criterio voraz, justificar	10
Greedy Choice	5
Inductive Substructure	5
Optimal Substructure	5
Conclusión	5

Primer Certamen
Algoritmos y Complejidad

7 de mayo de 2022



http://www.xkcd.org/2605

La entrega es en páginas separadas por pregunta, cada una debidamente identificada (nombre, rol, certamen y ramo). Si no responde, entregue una hoja en blanco.

1. Las notaciones asintóticas de Vinogradov se definen como sigue:

   $f(x) \ll g(x)$ $\qquad$ $f(x) = O(g(x))$

   $f(x) \gg g(x)$ $\qquad$ $g(x) = O(f(x))$

   $f(x) \asymp g(x)$ $\qquad$ $f(x) \ll g(x)$ y $f(x) \gg g(x)$

   Una relación de orden R es:

   Reflexiva: $a\ R\ a$ para todo a

   Transitiva: $a\ R\ b$ y $b\ R\ c$ implica $a\ R\ c$ para todo a, b, c

   Antisimétrica: $a\ R\ b$ y $b\ R\ a$ implica $a = b$ para todo a, b

   ¿Cuáles de estas propiedades cumple $\ll$, que claramente se entiende como una especie de «menor que»?

   (20 puntos)

2. Sea $f(x)$ con un cero de multiplicidad m en x^*, o sea, $f(x) = (x - x^*)^m g(x)$ con $g(x^*) \neq 0$. El *método de Newton modificado* usa la iteración:

   $$x_{n+1} = x_n - \alpha \frac{f(x_n)}{f'(x_n)}$$

   ¿Cuál es el mejor valor de α en este caso? ¿Cuál es el orden de convergencia resultante?

   (25 puntos)

3. La UTFSM decide instalar un nuevo edificio, la Torre Laudien, para acomodar a sus muchos estudiantes de fuera de la región. Hay que considerar medidas especiales para los pisos desde los cuales si cae un ~~estudiante copeteado~~ huevo éste se quiebra. Obviamente, si cae de un piso y se quiebra, se quebrará al caer de los pisos superiores también. Interesa identificar el piso K que es el más bajo tal que un huevo dejado caer de él se quiebra. Demuestre que si cuenta con un único huevo se requiere un mínimo de K intentos para determinar K.

   (20 puntos)

4. Un autómata finito puede entenderse como un digrafo, con los estados como vértices y las transiciones como arcos. Estados desde los que no se pueden alcanzar estados finales o que no son alcanzables desde el estado inicial claramente son inútiles. Plantee un algoritmo para determinar los estados útiles.

(30 puntos)

5. Describa cómo usar *backtracking* para hallar el número de caminos desde la esquina superior izquierda a la inferior derecha en una matriz de $m \times n$, si en cada paso puede moverse a la derecha o hacia abajo. El algoritmo 1 es nuestro esquema general. Explique las estructuras de datos empleadas, en particular cómo efectúa hacer/deshacer movidas.

Algoritmo 1: Esquema de *backtracking*

```
procedure backtrack(a, k)
    if a is solution then
        Process a
    end
    k ← k + 1
    Compute S_k
    while S_k ≠ ∅ do
        a_k ← select an element of S_k
        S_k ← S_k ∖ {a_k}
        Register move a_k
        backtrack(a, k)
        Undo move a_k
    end
end
```

(30 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

7 de mayo de 2022

1. La definición básica a usar (estamos suponiendo que el caso de interés es $x \to \infty$) es:

   $$f(x) = O(g(x))$$

   si hay constantes $c > 0$ y x_0 tales que siempre que $x \geq x_0$ se cumple:

   $$|f(x)| \leq c|g(x)|$$

   Las propiedades de relaciones de orden son:

   Reflexiva: Se cumple $f(x) \ll f(x)$, ya que $f(x) = O(f(x))$ es obvio de la definición (sirve x_0 arbitrario, $c = 1$).

   Transitiva: Si $f(x) \ll g(x)$ y $g(x) \ll h(x)$ es que hay constantes x_1, x_2 y c_1, c_2 tales que:

   $$\begin{aligned} |f(x)| &\leq c_1|g(x)| && x \geq x_1 \\ |g(x)| &\leq c_2|h(x)| && x \geq x_2 \end{aligned}$$

   Concluimos que si $x \geq \text{máx}\{x_1, x_2\}$:

   $$|f(x)| \leq c_1 c_2 |h(x)|$$

   que es $f(x) \ll h(x)$. Siempre se cumple.

   Antisimétrica: No siempre se cumple. Por ejemplo:

   $$\begin{aligned} 3x &\ll x \\ x &\ll 3x \end{aligned}$$

   pero $3x \neq x$.

   Es obvia entonces la pregunta de si $\asymp$ es una relación de equivalencia, con la que podríamos salvar lo anterior (exija $\asymp$ en vez de $=$ en antisimetría). Veamos:

   Reflexiva: Es claro que $f(x) \asymp f(x)$.

Transitiva: Como $f(x) \asymp g(x)$ si y solo si tanto $f(x) \ll g(x)$ como $g(x) \ll f(x)$, y vimos antes que $\ll$ es transitiva, también lo es $\asymp$.

Simétrica: Nuevamente $f(x) \asymp g(x)$ si y solo si tanto $f(x) \ll g(x)$ como $g(x) \ll f(x)$, que es lo mismo que $g(x) \asymp f(x)$. Se cumple.

O sea, $\ll, \gg, \asymp$ sí forman un juego de relaciones afín a $\leq, \geq, =$.

Puntajes

Total		20
Reflexiva	6	
Transitiva	6	
Antisimétrica	6	
Conclusión	2	

2. La iteración de Newton es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

sabemos que:

$$\begin{aligned} f(x) &= (x-x^*)^m g(x) \\ f'(x) &= m(x-x^*)^{m-1} g(x) + (x-x^*)^m g'(x) \\ &= (x-x^*)^{m-1}(mg(x) + (x-x^*)g'(x)) \end{aligned}$$

Expandiendo por Taylor alrededor de x^*:

$$\begin{aligned} g(x^*+e) &= g(x^*) + g'(x^*)e + O(e^2) \\ g'(x^*+e) &= g'(x^*) + g''(x^*)e + O(e^2) \end{aligned}$$

Con esto, definiendo $M = g'(x^*)/g(x^*)$:

$$\begin{aligned} f(x^*+e) &= e^m(g(x^*) + g'(x^*)e + O(e^2)) \\ &= e^m g(x^*)(1 + Me + O(e^2)) \\ f'(x^*+e) &= e^{m-1}(mg(x^*) + mg'(x^*)e + g'(x^*)e + g''(x^*)e^2 + O(e^2)) \\ &= e^{m-1} mg(x^*)\left(1 + \frac{(m+1)M}{m}e + O(e^2)\right) \end{aligned}$$

Podemos desarrollar para el error de Newton modificado:

$$\begin{aligned} e_{n+1} &= e_n - \alpha \frac{e_n^m g(x^*)(1 + Me_n + O(e_n^2))}{m} e_n^{m-1} g(x^*)(1 + (m+1)M/me_n + O(e_n^2)) \\ &= e_n - \frac{\alpha e_n}{m}\left(1 - \frac{M}{m}e_n + O(e_n^2)\right) \end{aligned}$$

(En realidad basta la estimación obvia $1 + O(e_n)$ del paréntesis para nuestros efectos.) Se ve que el método es cuadrático si $\alpha = m$, simplificando lo anterior resulta:

$$\begin{aligned} e_{n+1} &= \frac{M}{m}e_n^2 + O(e_n^3) \\ &= \frac{g'(x^*)}{mg(x^*))}e_n^2 + o(e_n^2) \end{aligned}$$

Puntajes

Total	25
Expresar $g(x)$, $g'(x)$	5
Plantear estimación para e_{n+1}	5
Simplificar	10
Conclusión	5

3. Por contradicción. Suponga que no se intentan todos los pisos, digamos que el lanzamiento más alto en que el huevo no se quiebra es desde el piso r y se quiebra al lanzarse del piso $s > r+1$. Con esto solo sabemos $r < K \leq s$, no queda determinado K. Un adversario puede arreglar que el huevo se quiebre la primera vez que omitimos un piso.

Puntajes

Total	30
Razonamiento claro y convincente	30

4. Nuestra notación gráfica para autómatas finitos es precisamente como grafos dirigidos rotulados. Podemos identificar los estados alcanzables desde el estado inicial iniciando un recorrido «hacia adelante» desde el estado inicial, eliminando luego los estados que no pueden alcanzarse así. Podemos luego iniciar recorridos «hacia atrás» desde los estados finales para identificar los estados desde los que se llega a estados finales, eliminando los que no llevan a estados finales quedan los estados que se visitan al procesar alguna palabra del lenguaje. Habrá que complementar con un estado muerto de ser necesario (es inútil según lo anterior).

Puntajes

Total		30
Razonamiento claro y convincente	15	
Usar recorrido de grafos	15	

5. Complementamos el esqueleto dado. Datos globales dan la posición actual. Para hacer o deshacer una movida basta aumentar o disminuir la coordenada del caso, siempre cuidando de no salir del arreglo. Terminamos (tenemos una solución) si llegamos al final. Procesar una solución es simplemente contabilizarla. El listado 1 da la búsqueda recursiva,

```
void bt()
{
    if(x == 1 && y == 1)
        ++paths;

    if(x > 1) {
        x--; bt(); x++;
    }
    if(y > 1) {
        y--; bt(); y++;
    }
}
```

Listado 1: La función bt()

el listado 2 es el programa principal (variables globales y programa mismo).

```
int x, y; /* Current position */
int paths;
int main(int argc, char *argv[])
{
    if(argc != 3) {
        fprintf(stderr, "usage: %s m n\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    x = atoi(argv[1]); y = atoi(argv[2]);
    paths = 0;
    bt();
    printf("For a matrix of %d by %d"
           " there are %d paths\n",
           x, y, paths);
    exit(EXIT_SUCCESS);
}
```

Listado 2: El programa principal

El problema en realidad es ridículo, hay $m+n-2$ movidas en total, de las que debemos elegir $m-1$ posiciones para las movidas hacia abajo. O sea, la respuesta es simplemente:

$$\binom{m+n-2}{m-1}$$

Igual resulta interesante evaluar el rendimiento de este algoritmo. Viendo el programa 1, es claro que el trabajo de cada llamada a `bt()` es relativamente constante, con lo que el número de llamadas es una medida adecuada. Llamemos $C(r,s)$ al número de llamadas para una matriz de $r \times s$. Tenemos la recurrencia:

$$C(r+1,s+1) = 1 + C(r,s+1) + C(r+1,s) \tag{1}$$

Condición de contorno es $C(1,1) = 1$.

De la recurrencia vemos que $C(k,1) = C(1,k) = k-1$ para todo k. Resulta que podemos poner condiciones:

$$C(k,0) = C(0,k) = 0 \text{para todo } k \tag{2}$$

consistentes con lo anterior. Estas son más cómodas de manejar en lo que sigue.

Para resolver la recurrencia (1) usamos funciones generatrices. Definimos:

$$g(x,y) = \sum_{r,s\geq 0} C(r,s)x^r y^s \tag{3}$$

Multiplicando (1) por $x^r y^s$ y sumando obtenemos:

$$\sum_{r,s\geq 0} C(r+1,s+1)x^r y^s = \sum_{r,s\geq 0} x^r y^s + \sum_{r,s\geq 0} C(r+1,s)x^r y^s + \sum_{r,s\geq 0} C(r,s+1)x^r y^s$$

Reconocemos las sumas resultantes, usamos las condiciones (2):

$$\begin{aligned}
\sum_{r,s\geq 0} C(r+1,s+1)x^r y^s &= \frac{g(x,y) - \sum_{r\geq 0} C(r,0)x^r - \sum_{s\geq 0} C(0,s)y^s + C(0,0)}{xy} \\
&= \frac{g(x,y)}{xy} \\
\sum_{r,s\geq 0} x^r y^s &= \left(\sum_{r\geq 0} x^r\right)\cdot\left(\sum_{s\geq 0} y^s\right) \\
&= \frac{1}{(1-x)(1-y)} \\
\sum_{r,s\geq 0} C(r+1,s)x^r y^s &= \frac{g(x,y) - \sum_{s\geq 0} C(0,s)y^s}{x} \\
&= \frac{g(x,y)}{x} \\
\sum_{r,s\geq 0} C(r,s+1)x^r y^s &= \frac{g(x,y) - \sum_{r\geq 0} C(r,0)x^r}{y} \\
&= \frac{g(x,y)}{y}
\end{aligned}$$

Juntando las piezas:

$$\frac{g(x,y)}{xy} = \frac{1}{(1-x)(1-y)} + \frac{g(x,y)}{x} + \frac{(g(x,y)}{y}$$

Despejando:

$$g(x,y) = \frac{xy}{(1-x)(1-y)(1-x-y)} \tag{4}$$

Agradable simetría, pero enfrentar esto directamente se ve desalentador. Dividimos en fracciones parciales en x, en la esperanza de obtener algo más manejable:

$$g(x,y) = \frac{1}{1-x-y} - \frac{1}{(1-x)(1-y)} \tag{5}$$

La segunda fracción es conocida, expandir la primera como serie geométrica da:

$$\begin{aligned}\frac{1}{1-x-y} &= \sum_{n\geq 0}(x+y)^n \\ &= \sum_{n\geq 0}\sum_{0\leq k\leq n}\binom{n}{k}x^k y^{n-k} \\ &= \sum_{r,s\geq 0}\binom{r+s}{r}x^r y^s\end{aligned}$$

Uniendo las anteriores y extrayendo coeficientes:

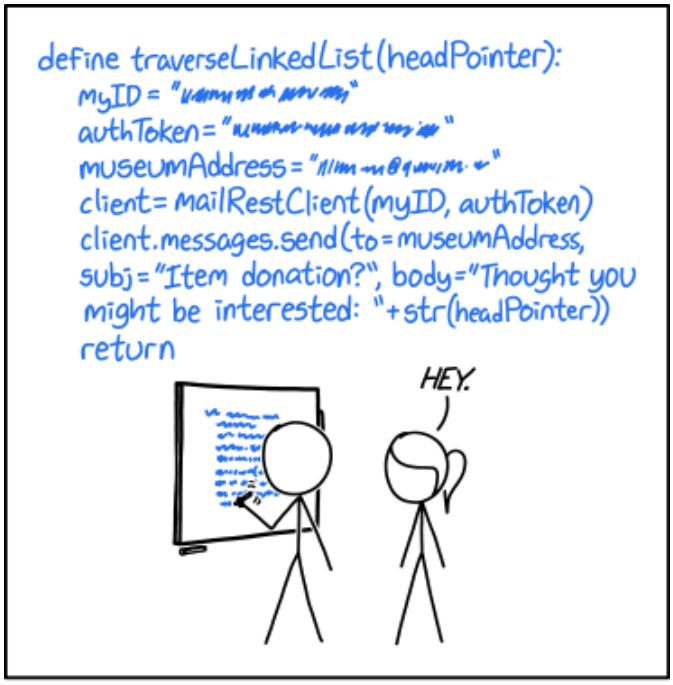
$$\begin{aligned}C(r,s) &= [x^r y^s]\,g(x,y) \\ &= \binom{r+s}{r} - 1\end{aligned}$$

Puntajes

Total		30
Datos globales	10	
Hacer/deshacer movidas	10	
Condición de fin	10	

Primer Certamen
Algoritmos y Complejidad

29 de octubre de 2021



CODING INTERVIEW TIP: INTERVIEWERS GET REALLY MAD WHEN YOU TRY TO DONATE THEIR LINKED LISTS TO A TECHNOLOGY MUSEUM.

http://www.xkcd.org/2483

La entrega es en formato electrónico por Aula (PDF, odt, doc, docx). Por favor, una respuesta por página. No se aceptan páginas escritas a mano escaneadas. Nos reservamos el derecho de interrogar sobre lo entregado a un conjunto de estudiantes.

Indique en la portada de su entrega electrónica el tiempo que le tomó el desarrollo (en minutos). Registre cualquier recurso externo que haya utilizado en el desarrollo de la prueba.

Indique el sistema de álgebra simbólica que usa para apoyar sus desarrollos, dando los comandos principales que emplea.

1. Confirme o desmienta: $f(x) = O(g(x))$ si y solo si

$$\lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| \neq \infty$$

(15 puntos)

2. Proponga una manera eficiente de calcular $\ln(x+1) - \ln x$ para x grande. Obtenga el valor del caso con 3 cifras significativas si $x = 20$.

(15 puntos)

3. Para la ecuación $x^3 = 0{,}5$ podemos definir iteraciones de la forma $x_{n+1} = g(x_n)$. Considerando las alternativas:

$$g_a(x) = \frac{1}{2x^2}$$
$$g_b(x) = \frac{x}{2} + \frac{1}{4x^2}$$
$$g_c(x) = \frac{2x}{3} + \frac{1}{6x^2}$$

   a) Demuestre que las tres opciones tienen $\sqrt[3]{0{,}5}$ como punto fijo.

   b) ¿Cuáles de las tres convergen? ¿Cuál converge más rápido cerca del punto fijo?

(25 puntos)

4. Se desea confeccionar una tabla de valores de la función:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-u^2}\,\mathrm{d}u$$

   con valores igualmente espaciados en $h = a \cdot 10^{-n}$, donde para comodidad a es 1, 2 o 5, de forma que interpolando linealmente obtenemos valores con 5 cifras decimales correctas (5 cifras después de la coma) en el rango 0 a 10. ¿Cuántos decimales correctos entrega esta tabla si usamos interpolación cúbica? Se sabe que $0 \le \operatorname{erf}(x) < 1$ para $x \ge 0$.

(30 puntos)

5. Se da un *string* que representa una palabra binaria con comodines `?`, que pueden reemplazarse por `0` o `1`. Por ejemplo, el patrón `01?1?` representa las palabras `01010`, `01011`, `01110`, `01111`.

   Usando *backtracking* escriba una función que tome un patrón como descrito y retorne todas las palabras que coinciden con el patrón.

(30 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

29 de octubre de 2021

1. La definición del límite finito:

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right| = a$$

es que para todo $\varepsilon > 0$ hay x_0 tal que si $x \geq x_0$ es:

$$\left|\left|\frac{f(x)}{g(x)}\right| - a\right| \leq \varepsilon$$

Debemos demostrar si y solo si, demostramos implicancia en ambas direcciones. Primero, supongamos que el límite existe y es finito. Podemos reorganizar la definición del límite:

$$\left|\left|\frac{f(x)}{g(x)}\right| - a\right| \leq \varepsilon$$
$$||f(x)| - a|g(x)|| \leq |g(x)|$$

Esto implica:

$$|f(x)| - |a| \cdot |g(x)| \leq \varepsilon|g(x)|$$
$$|f(x)| \leq |g(x)| \cdot (\varepsilon + |a|)$$

Esto corresponde a la definición de $f(x) = O(g(x)$.

Pero también es:

$$|f(x)| \leq \varepsilon|g(x)| + a|g(x)|$$
$$|f(x)| \geq |g(x)| \cdot (\varepsilon + |a|)$$

Siempre que $a \neq 0$, esto es $f(x) = \Omega(g(x)$, con lo que concluimos que si el límite no es cero, es $f(x) = \Theta(g(x))$.

En la dirección contraria, supongamos que $f(x) = O(g(x))$. Esto significa que hay x_0 y $C > 0$ tales que para todo $x \geq x_0$:

$$|f(x)| \leq C|g(x)|$$

Esto solo dice que:

$$\left|\frac{f(x)}{g(x)}\right| \leq C$$

No se ve demasiado prometedor. En particular, consideremos:

$$f(x) = x(1+\sin x)$$
$$g(x) = x$$

Es claro que $f(x) = O(g(x))$ (con $x_0 = 0$ y $C = 2$), sin embargo el límite no existe.

Concluimos que la implicancia solo vale en una dirección. Esto nos da una manera de demostrar $O(\cdot)$ y $\Theta(g(x))$ que puede ser útil, claro está.

Puntajes

Total	15
Definición de $O(\cdot)$, límite	4
Comparar ambas	6
Conclusiones	5

2. Tenemos:

$$\ln(x+1) - \ln x = \ln\left(1 + \frac{1}{x}\right)$$

Conocemos la serie de potencias, válida para $|u| < 1$:

$$\ln(1+u) = \sum_{k\geq 1} \frac{(-1)^k u^k}{k}$$

Para $u > 0$ esta es una serie alternante, el error al truncar es menor que el primer término omitido. Para $x = 20$ es $u = 1/x = 0{,}1$, del primer término de la serie sabemos que el valor de $\ln 21 - \ln 20$ es aproximadamente $1/20 = 0{,}500\,00$, tres cifras significativas son un error relativo de $0{,}5 \times 10^{-3}$. Necesitamos determinar el número de términos n tal que:

$$\begin{aligned}
\frac{20^{-n}}{n} &\leq 20^{-1} \cdot 0{,}5 \times 10^{-3} \\
&= 0{,}000\,002\,5 \\
20^n \cdot n &\geq 4000{,}000\,00
\end{aligned}$$

Esto no tiene solución elemental, pero la iteración de punto fijo siguiente se ve promisoria:

$$\begin{aligned}
20^n \cdot n &= 4000 \\
n \ln 20 + \ln n &= \ln 40000 \\
n &= \frac{\ln 4000 - \ln n}{\ln 20}
\end{aligned}$$

El programa `20212c1p2.py` (vea el listado 1) entrega $n = 2{,}476\,12$, con $n = 3$ debiéramos tener de sobra:

$$20 \cdot \frac{20^{-3}}{3} = 0{,}000\,833$$

```
#!/usr/bin/env python

from math import *

n0 = 3
while True:
    n1 = (log(4000) - log(n0)) / log(20);
    if abs(n1 - n0) < 0.1:
        break
    n0 = n1

print(f"n = {n1} ({n1 - n0})")
```

Listado 1: Cálculo de n

Vale decir, el valor buscado se obtiene con dos términos:

$$\begin{aligned}\ln 21 - \ln 20 &= \frac{1}{20} - \frac{1}{2\cdot 20^2} \\ &= 0{,}048\,75\end{aligned}$$

Para referencia, el valor correcto es 0,048 790 16. En la práctica probablemente usaríamos un programa como `20212c1p2x.py` para calcular $\ln(1+u)$, la función central es el listado 2, que calcula los términos sucesivos hasta que el término resulta menor que el error solicitado (lo que es simple de calcular en este caso), aunque para mitigar acumulación de errores de redondeo sumaríamos desde el menor término al mayor, estimando (como antes) el número de términos requeridos para luego efectuar la suma.

```
def ln1(u, eps):
    sum = 0; k = 1; pow = u
    while True:
        term = pow / k
        sum += term
        if abs(term) < sum * eps:
            break
        pow *= -u; k += 1
    return sum
```

Listado 2: Cálculo de $\ln(1+x) - \ln x$

Puntajes

Total	15
Serie para $\ln(1+x)$	5
Número de términos requerido	5
Cálculo del valor	5

3. Vamos por turno.

   a) Veamos cada una de las funciones propuestas en el punto fijo $x^* = \sqrt[3]{0{,}500\,00}$.

$$\begin{aligned} g_a(x^*) &= \frac{1}{2(\sqrt[3]{1/2})^2} \\ &= \frac{2^{2/3}}{2} \\ &= 2^{-1/3} \\ &= x^* \end{aligned}$$

   Es punto fijo.

$$\begin{aligned} g_b(x^*) &= \frac{\sqrt[3]{1/2}}{2} + \frac{1}{4(\sqrt[3]{1/2})^2} \\ &= 2^{-1-1/3} + 2^{-2+2/3} \\ &= 2 \cdot 2^{-4/3} \\ &= 2^{-1/3} \\ &= x^* \end{aligned}$$

   También es punto fijo.

$$\begin{aligned} g_c(x^*) &= \frac{2\sqrt[3]{1/2}}{3} + \frac{1}{6(\sqrt[3]{1/2})^2} \\ &= \frac{2^{1-1/3}}{3} + \frac{2^{-1+2/3}}{3} \\ &= \frac{2^{2/3}}{3} + \frac{2^{-1/3}}{3} \\ &= \frac{2+1}{3} \cdot 2^{-1/3} \\ &= x^* \end{aligned}$$

   Nuevamente es punto fijo.

   b) Sabemos que cerca del punto fijo x^* la iteración converge si y solo si $|g'(x^*)| < 1$. Además, el análisis de convergencia indica que mientras me-

nor es $|g(x^*)|$ más rápida es la convergencia. Veamos:

$$g_a'(x) = -\frac{1}{x^3}$$
$$g_a'(x^*) = -2$$
$$g_b'(x) = \frac{1}{2} - \frac{1}{2x^3}$$
$$g_b'(x^*) = -\frac{1}{2}$$
$$g_c'(x) = \frac{2}{3} - \frac{1}{3x^3}$$
$$g_c'(x^*) = 0$$

Concluimos que la iteración g_a no converge, que g_b converge linealmente y que g_c converge más que linealmente. Resulta que g_c es la iteración de Newton (converge cuadráticamente).

El programa `20212c1p3.py` (listado 3) calcula los valores indicados.

```
#!/usr/bin/env python

from sympy import *

x = symbols('x')

def ga(x):
    return 1 / (2 * x ** 2)

def gb(x):
    return x / 2 + 1 / (4 * x ** 2)

def gc(x):
    return 2 * x / 3 + 1 / (6 * x ** 2)

xs = sympify("2 ** (-1/3)")

print(f"ga(xs) = {simplify(ga(xs))}")
print(f"gb(xs) = {simplify(gb(xs))}")
print(f"gc(xs) = {simplify(gc(xs))}")
print()

def gap(x):
    return diff(ga(x))

def gbp(x):
    return diff(gb(x))
```

```python
def gcp(x):
    return diff(gc(x))

print(f"ga'(xs) = {diff(ga(x), x).subs(x, xs)}")
print(f"gb'(xs) = {diff(gb(x), x).subs(x, xs)}")
print(f"gc'(xs) = {diff(gc(x), x).subs(x, xs)}")
print()

def f(x):
    return x ** 3 - Rational(1, 2)

print(f"Newton: {simplify(x - f(x) / diff(f(x), x))}")
```

Listado 3: Determinar convergencia

Puntajes

Total		25
a) Es punto fijo		6
g_a	2	
g_b	2	
g_c	2	
b) Convergencia		19
Criterio de $g'(x^*)$	5	
$g'_a(x^*)$	3	
$g'_b(x^*)$	3	
$g'_c(x^*)$	3	
Más rápido de los tres	2	

4. Por el teorema que da el error de interpolación de un polinomio lineal sabemos que el error al interpolar $f(x)$ entre x_0, x_1 linealmente cumple, con algún $x_0 \leq \xi \leq x_1$:

$$E = \frac{f''(\xi)}{2!}(x - x_0)(x - x_1)$$

Podemos acotar el error como:

$$\begin{aligned} |E| &= \left| \frac{f''(\xi)}{2!}(x - x_0)(x - x_1) \right| \\ &\leq \frac{1}{2} \max_{x_0 \leq \xi \leq x_1} \left\{ f''(\xi) \right\} \cdot \max_{x_0 \leq x \leq x_1} \left\{ (x - x_0)(x - x_1) \right\} \\ &= \frac{1}{2} \max_{x_0 \leq \xi \leq x_1} \left\{ f''(\xi) \right\} \cdot \frac{(x_1 - x_0)^2}{8} \end{aligned}$$

En realidad, nos interesa el error sobre todo el rango, analicemos la segunda derivada de erf:

$$\begin{aligned} \operatorname{erf}(x) &= \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-u^2} \, \mathrm{d}u \\ \operatorname{erf}''(x) &= -\frac{4x}{\sqrt{pi}} \mathrm{e}^{-x^2} \\ \operatorname{erf}'''(x) &= \frac{4(2x^2 - 1)}{\sqrt{pi}} \mathrm{e}^{-x^2} \\ \operatorname{erf}^{(iv)}(x) &= \frac{8(4x^4 - 12x^2 + 3)}{\sqrt{pi}} \mathrm{e}^{-x^2} \end{aligned}$$

Para obtener el máximo del valor absoluto de la segunda derivada de la función a interpolar, debe ser $\operatorname{erf}'''(x^*) = 0$, lo que da $2(x^*)^2 - 1 = 0$, lo que da $x^* = \sqrt{1/2}$. Como $\operatorname{erf}^{(iv)}(x^*)$ es negativo, es el mínimo. El mínimo es:

$$\begin{aligned} \operatorname{erf}''(x^*) &= -\frac{2}{\sqrt{\pi}} \mathrm{e}^{-1/2} \\ &= -0{,}967\,88 \end{aligned}$$

O sea, una sobreestimación del error en el rango $[0, \infty)$ es:

$$\begin{aligned} |E| &\leq \frac{h^2}{16} \cdot \frac{2}{\sqrt{\pi}} \mathrm{e}^{-1/2} \\ &= 0{,}060\,49 h^2 \end{aligned}$$

Queremos 5,000 00 cifras decimales, lo que es $|E| \leq 0{,}5 \times 10^{-5}$:

$$\begin{aligned} 0{,}5 \times 10^{-5} &= 0{,}060\,49 h^2 \\ h &= 0{,}009\,09 \end{aligned}$$

Con las restricciones indicadas, debiéramos listar valores cada 0,005 (casi estamos con 0,01, probablemente baste con este valor, nuestras estimaciones son un tanto sueltas).

Las expresiones y valores citados antes da el programa `20212c1p4.py`, listado 4.

```
#!/usr/bin/env python

from sympy import *

x = symbols('x')

def erf2(x):
    return diff(2 * exp(-x**2) / sqrt(pi), x)

def erf3(x):
    return diff(2 * exp(-x**2) / sqrt(pi), x, 2)

def erf4(x):
    return diff(2 * exp(-x**2) / sqrt(pi), x, 4)

print(f"Third derivative: {erf3(x)}")

xs = sqrt(Rational(1, 2))

print(f"At x^* = {xs}:")
print("Fourth derivative: {erf4(x).subs(x, xs)}",
      f"({float(erf4(x).subs(x, xs))})")
print("Second derivative: {erf2(x).subs(x, xs)}",
      f"({float(erf2(x).subs(x, xs))})")
```

Listado 4: Para tabla de erf

Puntajes

Total	30
Error máximo sobre el rango	15
Derivadas necesarias	10
Resultado	5

5. La solución recursiva es simple, programa `20212c1p5.py` (el listado 5 es la parte central).

```
def backtrack(k):
    global pattern, a, solutions

    if k == len(pattern):
        solutions += [''.join(a)]
    elif pattern[k] != '?':
        a[k] = pattern[k]
        backtrack(k + 1)
    else: # pattern[k] == '?'
        for s in ['0', '1']:
            a[k] = s
            backtrack(k + 1)

def matching(mpattern):
    global pattern, a, solutions

    pattern = list(mpattern)
    a = list(mpattern)
    solutions = []
    backtrack(0)
    return solutions
```

Listado 5: Expandir patrones

La función backtrack está tomada de nuestra función genérica, claro que usamos un arreglo global a en vez de un argumento. Al llegar al final agregamos la solución hallada al arreglo global `solutions`. Avanzamos a través del patrón `pattern`, si la posición actual es `'?'` tenemos las opciones `'0'` y `'1'`, en caso contrario solo el símbolo en esa posición. La función `matching` recibe el patrón, inicializa variables globales e invoca a backtrack, retornando el arreglo de resultados.

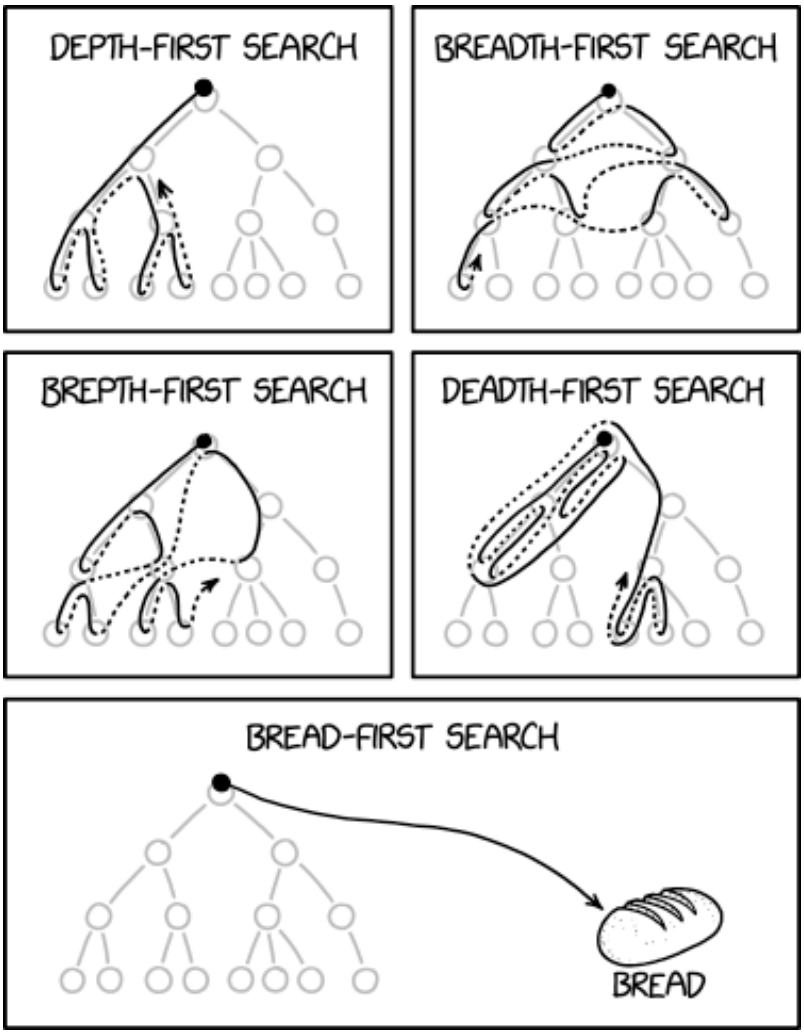
Puntajes

Total	30
Explicación del algoritmo	10
Backtracking	15
Código claro, limpio	5

HvB/LaTeX2ε

Primer Certamen
Algoritmos y Complejidad

28 de mayo de 2021



La entrega es en formato electrónico por Aula (PDF, odt, doc, docx). Por favor, una respuesta por página. No se aceptan páginas escritas a mano escaneadas. Nos reservamos el derecho de interrogar sobre lo entregado a un conjunto de estudiantes.

Indique en la portada de su entrega electrónica el tiempo que le tomó el desarrollo (en minutos). Registre cualquier recurso externo que haya utilizado en el desarrollo de la prueba.

1. Analice la convergencia del método de Newton a x^* bajo la condición que $f'(x^*) \neq 0$ y $f''(x^*) = 0$. ¿Cuál es el orden de convergencia?

   (25 puntos)

2. Determine hasta qué grado de polinomio es exacta la regla de cuadratura:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \frac{2}{3}\left(f\left(\frac{-\sqrt{2}}{2}\right) + f(0) + f\left(\frac{\sqrt{2}}{2}\right)\right)$$

   (35 puntos)

3. Se da una lista de n elementos. Considerando solamente la operación de comparar elementos (llamémosla $a \Leftrightarrow b$ que retorna si $a < b$, $a = b$ o $a > b$) considere las siguientes:

   a) Demuestre que se requieren al menos $n-1$ comparaciones para determinar si todos los elementos son iguales. Proponga un algoritmo que resuelva el problema con este número de comparaciones.

   b) Demuestre que se requieren al menos $n-1$ comparaciones para determinar si todos los elementos son distintos. Plantee un algoritmo que resuelva este problema. ¿Cuántas comparaciones efectúa su algoritmo?

   (30 puntos)

4. Sabemos que el problema INDEPENDENT SET (dado un grafo G y un entero k, ¿hay un conjunto de k vértices de G sin arcos entre ellos?) es NP-completo, por lo que un algoritmo eficiente que encuentre un máximo conjunto de vértices sin conexiones entre ellos es muy poco probable. Una solución de fuerza bruta es considerar todos los subconjuntos de vértices y registrar el de mayor tamaño que se encuentre.

   Escriba un programa C++ que use la estrategia esbozada para hallar un *independent set* máximo de un grafo. Como partida, se provee código C++ que lee y escribe grafos, y determina si un conjunto de vértices es independiente. Un programa ejemplo completo está en el directorio `P4`, para su solución debe completar el archivo `search.cc` del directorio `Sol`. Entregue una breve explicación de su algoritmo y su versión de `search.cc`.

   (35 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

28 de mayo de 2021

1. Expandiendo f y f' por Taylor alrededor del cero x^*, bajo el entendido $f(x^*) = f''(x^*) = 0$ y $f'(x^*) \neq 0$, tenemos:

$$\begin{aligned} f(x^*+e) &= f(x^*) + f'(x^*)e + \frac{1}{2}f''(x^*)e^2 + \frac{1}{6}f'''(x^*)e^3 + O(e^4) \\ &= f'(x^*)e + \frac{1}{6}f'''(x^*)e^3 + O(e^4) \\ f'(x^*+e) &= f'(x^*) + f''(x^*)e + \frac{1}{2}f'''(x^*)e^2 + O(e^3) \\ &= f'(x^*) + \frac{1}{2}f'''(x^*)e^2 + O(e^3) \end{aligned}$$

(Siguiendo la pista del desarrollo en clases, tomamos tres términos como primer intento.) Escribiremos las anteriores como:

$$\begin{aligned} f(x^*+e) &= f'(x^*)e(1 + Me^2 + O(e^3)) \\ f'(x^*+e) &= f'(x^*)(1 + 3Me^2 + O(e^3)) \end{aligned}$$

donde:

$$M = \frac{f'''(x^*)}{6f'(x^*)}$$

La iteración de Newton es, con $e_n = x_n - x^*$:

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ e_{n+1} &= e_n - \frac{f(x^*+e_n)}{f'(x^*+e_n)} \end{aligned}$$

Simplifiquemos la fracción:

$$\begin{aligned} \frac{f'(x^*)e_n(1 + Me_n^2 + O(e_n^3))}{f'(x^*)(1 + 3Me_n^2 + O(e_n^3))} &= e_n \frac{1 + Me_n^2 + O(e_n^3)}{1 + 3Me_n^2 + O(e_n^3)} \\ &= e_n(1 + Me_n^2 + O(e_n^3))(1 - 3Me_n^2 + O(e_n^3)) \\ &= e_n(1 - 2Me_n^2 + O(e_n^4)) \end{aligned}$$

De la iteración tenemos entonces:

$$\begin{aligned} e_{n+1} &= e_n - e_n(1 - 2Me_n^2 + O(e_n^4)) \\ &= 2Me_n^3 + O(e_n^5) \end{aligned}$$

El método es cúbico.

Puntajes

Total		25
Expandir f, f' alrededor de x^*	5	
Plantear iteración de Newton en e_n	5	
Simplificar la expresión asintótica	10	
Conclusión	5	

2. La forma más simple de responder esta pregunta es intentar con los polinomios $1, x, x^2, \ldots$ hasta que la fórmula no resulte exacta. Sabemos que:

$$\int_{-1}^{1} x^n \,\mathrm{d}x = \left.\frac{x^{n+1}}{n+1}\right|_{-1}^{1} = \begin{cases} \frac{2}{n+1} & n \text{ par} \\ 0 & n \text{ impar} \end{cases}$$

Por simetría, para potencias impares es exacta. Para potencias pares $n = 2k$:

- Para $k = 0$:

$$\begin{aligned} \frac{2}{3}(1+1+1) &= \frac{2}{3} \cdot 3 \\ &= 2 \end{aligned}$$

- Para $k > 0$:

$$\begin{aligned} \frac{2}{3}\left(\left(-\frac{\sqrt{2}}{2}\right)^{2k} + \left(-\frac{\sqrt{2}}{2}\right)^{2k}\right) &= \frac{4}{3} \cdot \left(\frac{\sqrt{2}}{2}\right)^{2k} \\ &= \frac{4}{3} \cdot \frac{2^k}{2^{2k}} \\ &= \frac{2^{n+2}}{3 \cdot 2^{2k}} \\ &= \frac{1}{3} \cdot 2^{2-k} \end{aligned}$$

Vemos que para que sea exacta debe ser $2 = 2^{2-k}$ y $2k+1 = 3$, o sea $k = 1$ ($n = 2$). Revisemos el siguiente valor, $k = 2$ ($n = 4$:

$$\begin{aligned} \frac{2}{3}\left(\left(\frac{\sqrt{2}}{2}\right)^4 + \left(\frac{\sqrt{2}}{2}\right)^4\right) &= \frac{1}{3} \cdot 2^{2-2} \\ &= \frac{1}{3} \\ \int_{-1}^{1} x^4 \,\mathrm{d}x &= \frac{2}{5} \end{aligned}$$

Como cabía esperar, no coinciden. La fórmula es exacta hasta $n = 3$.

Puntajes

Total	35
Bastan $\int_{-1}^{1} x^n \,\mathrm{d}x$	8
Para n impar es exacta por simetría	5
Plantear condición de máximo n par	10
Verificar que no funciona para $n+2$	7
Concluir máximo n	5

3. Son dos preguntas relacionadas.

   a) Si unimos cada par de elementos que algún algoritmo compara con arcos, tenemos un multigrafo de n vértices. Considerando solo algoritmos que no repiten comparaciones y no comparan elementos consigo mismos (ambos claramente desperdicio de operaciones), es un grafo. Para asegurar que se relacionan todos los elementos, el grafo debe ser conexo (si hay varias componentes conexas, sus vértices pueden tener valores diferentes y el algoritmo no tiene cómo darse cuenta).

   Un grafo conexo de n vértices con el mínimo número de arcos es un árbol, que tiene $n-1$ arcos.

   El algoritmo 1 obvio efectúa exactamente $n-1$ comparaciones en el peor caso (todos los elementos iguales), es óptimo.

Algoritmo 1: Determinar si todos son iguales

```
function all_equal(a)
    for k ← 2 to n do
        if a_1 ≠ a_k then
            return False
        end
    end
    return True
```

   b) Igual que en la respuesta 3*a*, son necesarias al menos $n-1$ comparaciones (si quedan elementos no conectados con los demás, pueden ser iguales a otros). El algoritmo 2 muestra que en el mejor caso (los elementos dados en orden creciente) se puede efectuar con $n-1$ comparaciones. Es claro que se puede responder con $n(n-1)/2$ comparaciones (comparar cada elemento con todos los demás, corresponde al grafo K_n), vea el algoritmo 4. Pero podemos hacerlo en $O(n \log n)$ comparacio-

Algoritmo 2: Determinar si todos son diferentes

```
function all_different(a)
    for k ← 1 to n − 1 do
        c ← a_k ⇔ a_{k+1}
        if c = Equal then
            return False
        else if c = Greater then
            return DontKnow
        end
    end
    return True
```

Algoritmo 3: Determinar si todos son diferentes, cada uno contra los demás

```
function all_different(a)
    sort(a)
    for i ← 1 to n − 1 do
        for j ← i + 1 to n do
            if a_i ⇔ a_j = Equal then
                return False
            end
        end
    end
    return True
```

nes: basta ordenar los elementos (requiere $O(n \log n)$ comparaciones) y luego verificar que no hay repetidos ($n-1$ comparaciones adicionales), algoritmo 3

Algoritmo 4: Determinar si todos son diferentes, ordenando

```
function all_different(a)
    sort(a)
    for k ← 1 to n − 1 do
        c ← a_k ⇔ a_{k+1}
        if c = Equal then
            return False
        end
    end
    return True
```

Puntajes

Total			30
a) Todos iguales		10	
Demostración	5		
Algoritmo	5		
b) Todos diferentes		20	
Demostración cota inferior	5		
Algoritmo propuesto	8		
Número de comparaciones de la propuesta	7		

4. Proponemos dos soluciones: generar ciegamente todos los subconjuntos de vértices y verificar si son independientes (`search-all.cc` en `P4`, esta es la idea esbozada en el enunciado), o usar el hecho que si un subconjunto de vértices no es independiente no tiene sentido agregar vértices (`search-cut.cc` en `P4`).

Puntajes

Total		35
Explicación del algoritmo	10	
Backtracking en `search.cc`	15	
Resultados correctos (cardinalidades)	5	
Código claro, limpio	5	

Primer Certamen
Algoritmos y Complejidad

6 de noviembre de 2020

DIALECT QUIZ

COMPARE ANSWERS WITH YOUR FRIENDS!

HOW DO YOU ADDRESS A GROUP OF TWO OR MORE PEOPLE?

A) YOU
B) Y'ALL
C) I HAVE NOT BEEN AROUND TWO OR MORE PEOPLE FOR SO LONG THAT I CAN'T REMEMBER

HOW DO YOU PRONOUNCE "PENELOPE"?

A) RHYMES WITH "ANTELOPE"
B) RHYMES WITH "DEVELOP"

WHAT DO YOU CALL THE SCIENTIFIC FIELD THAT STUDIES THE STARS?

A) ASTROLOGY
B) AGRONOMY
C) COSMETOLOGY

HOW DO YOU PRONOUNCE "GENRE"?

A) GONE-RA
B) JUH-NEER
C) JEN-ER-UH

DO YOU PRONOUNCE "GOOGLE" WITH A HIGH-PITCHED YELP ON THE...

A) FIRST SYLLABLE
B) SECOND SYLLABLE

WHAT DO YOU CALL THE THING ON THE WALL AT SCHOOL THAT YOU DRINK WATER FROM?

A) GUTTER PIPE
B) DRAINPIPE

HOW DO YOU PRONOUNCE THE NAME FOR A SHORT SILENT VIDEO FILE?

A) ANIMATED GIVE
B) ANIMATED GIFT

WHAT DO YOU CALL THE BASEBALL-SIZED GARDEN BUGS THAT, WHEN POKED, GLOW BRIGHTLY AND EMIT A WARBLING SCREAM?

A) WHAT?
B) LAWN BUDDIES

WHAT DO YOU CALL THE MISLEADING LINES PAINTED BY DISGRUNTLED HIGHWAY WORKERS TO TRICK CARS INTO DRIVING OFF THE ROAD?

A) PRANK LINES
B) DEVIL'S MARKS
C) FOOL-ME LINES
D) FAUXGUIDES
E) DELAWARE LINES

WHAT DO YOU CALL THE BLUE-GREEN PLANET IN THE OUTER SOLAR SYSTEM?

A) URANUS
B) NEPTUNE

WHAT DO YOU CALL THIS TOOL?

A) BANGER
B) NAIL AXE
C) WOOD MAGE WAND
D) I'M FAMILIAR WITH THIS TOOL BUT HAVE NO SPECIFIC WORD FOR IT
E) I HAVE NEVER SEEN IT BEFORE

WHAT DO YOU CALL A LONG SANDWICH WITH MEATS AND LETTUCE AND STUFF?

A) A LONG SANDWICH WITH MEATS AND LETTUCE AND STUFF
B) A LONGWICH
C) A SALAD HOT DOG

WHAT DO YOU CALL THE SCALY MANY-LEGGED ANIMAL OFTEN FOUND IN ATTICS?

A) LIGHTBULB EATER
B) I HAVE NO SPECIAL NAME FOR THEM
C) I'VE NEVER LOOKED IN MY ATTIC

WHAT DO YOU SAY WHEN SOMEONE AROUND YOU SNEEZES?

A) "WHAT WAS THAT?"
B) "OH, WOW."
C) [QUIETLY] "YIKES."

La entrega es en formato electrónico por Aula (PDF, odt, doc, docx). Por favor, una respuesta por página. No se aceptan páginas escritas a mano escaneadas. Nos reservamos el derecho de interrogar sobre lo entregado a un conjunto de estudiantes.

Indique en la portada de su entrega electrónica el tiempo que le tomó el desarrollo (en minutos). Registre cualquier recurso externo que haya utilizado en el desarrollo de la prueba.

0. Responda el quiz de la portada.

(0 puntos)

1. Las siguientes son independientes, puede usar la conclusión de una parte en las demás aún si no la responde.

   a) El algoritmo de Newton solo converge linealmente si el cero x^* de la función f que buscamos es múltiple (o sea, podemos escribir $f(x) = (x - x^*)^m g(x)$, con $m > 1$ y $g(x^*) \neq 0$). Demuestre esta aseveración.

   b) Demuestre que la función $f(x)/f'(x)$ solo tiene ceros simples.

   c) Plantee un método iterativo basado en la observación del punto 1*b*.

   d) ¿Recomendaría el método del punto 1*c* para uso general? Justifique.

(40 puntos)

2. Suponga que usted es un pequeño comerciante en un lejano país. Las monedas son de denominaciones $1 = c_0 < c_1 < \cdots < c_n$ con $n > 1$. Por orden del Benevolente Primer Ciudadano, al dar vuelto debe dar el mínimo número de monedas (no quiere que se gaste la Ilustre Cara que está estampada en ellas).

   a) Demuestre que si $c_k = b^k$ para $b > 1$, el algoritmo voraz de entregar lo más posible desde la moneda más grande a la menor entrega un óptimo.

   b) Si las monedas son de $1, 7, 10$, demuestre que el algoritmo voraz no siempre da un óptimo.

(35 puntos)

3. Los coeficientes binomiales se definen por la recurrencia:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

   con condiciones de borde:

$$\binom{n}{0} = \binom{n}{n} = 1$$

   a) Plantee un algoritmo que calcule $\binom{n}{k}$ usando esta recurrencia.

   b) ¿Cuántas llamadas efectúa su algoritmo al calcular $\binom{n}{k}$? Plantee las relaciones que determinan esta cantidad, no las resuelva.

   c) Es obvio que se efectúa múltiples veces la misma llamada al traducir la recurrencia a una función recursiva. Escriba una variante memoizada, y calcule el número de llamadas que efectúa estas versión. Obtenga una cota asintótica simple y precisa.

(25 puntos)

4. Aritmética alternativa:

   a) Considere el algoritmo aprendido en el colegio para multiplicar números. Si la unidad de costo es una operación entre números de un dígito (una suma o una multiplicación), ¿cuál es el costo aproximado de multiplicar dos números de N dígitos? Obtenga una cota asintótica simple y precisa.

   b) El *algoritmo del campesino ruso* funciona como sigue: por ejemplo, para multiplicar 13 por 238, escriba una tabla en que divide sucesivamente 13 por 2 y duplica 238 en paralelo. Descarta aquellos números de las segunda columna que acompañan un número par en la primera columna, y suma los restantes.

13	238
6	~~476~~
3	952
1	1 904

   El resultado es $238 + 952 + 1\,904 = 3\,094$.

   Formalice este método, y calcule cuántas sumas efectúa (duplicar un número es sumarlo consigo mismo) y cuántas veces divide por dos. Indique máximos y mínimos de las cantidades indicadas al multiplicar A por B en ese orden. Note que la unidad de costo es distinta a la de la pregunta anterior.

(35 puntos)

HvB/LaTeX 2ε

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

6 de noviembre de 2020

0. Las respuestas más creativas se mencionarán en el Aula.

Puntajes

Total	0
Divertir mucho a quien corrige	5

1. Suponemos una función f con un cero de multiplicidad m en x^*, vale decir si escribimos $f(x) = (x-x^*)^m g(x)$, $g(x^*) \neq 0$ y es finito.

   a) Tenemos:

   $$\begin{aligned} f'(x) &= m(x-x^*)^{m-1} g(x) + (x-x^*)^m g'(x) \\ &= (x-x^*)^{m-1}(mg(x) + (x-x^*)g'(x)) \end{aligned}$$

   Para Newton tenemos:

   $$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{(x_n-x^*)^m g(x_m)}{(x_n-x^*)^{m-1}(mg(x) + (x-x^*)g'(x))} \\ &= x_n - \frac{(x_n-x^*)g(x_m)}{mg(x) + (x-x^*)g'(x)} \end{aligned}$$

   En término de los errores, $e_n = x_n - x^*$:

   $$\begin{aligned} e_n n+1 &= e_n - \frac{e_n g(x^*+e_n)}{mg(x^*+e_n) + e_n g'(x^*+e_n)} \\ &= e_n \left(1 - \frac{g(x^*+e_n)}{mg(x^*+e_n) + e_n g'(x^*+e_n)}\right) \end{aligned}$$

   Esto hace sospechar que es de primer orden, con constante $1 - 1/m = (m-1)/m$. Mientras mayor sea m, peor.
   Para confirmar, expandimos g y g' por Taylor:

   $$\begin{aligned} g(x^*+e) &= g(x^*) + g'(x^*)e + O(e^2) \\ &= g(x^*)(1 + Me + O(e^2)) \\ g'(x^*+e) &= g'(x^*) + O(e) \end{aligned}$$

   Acá definimos $M = g(x^*)/g'(x^*)$. La fracción es:

   $$\begin{aligned} \frac{g(x^*+e)}{mg(x^*+e) + eg'(x^*+e)} &= \frac{g(x^*)(1+Me+O(e^2))}{mg(x^*)\left(1 + \frac{M}{m}e + O(e^2)\right)} \\ &= \frac{1}{m}\left(1 + Me + O(e^2)\right)\left(1 - \frac{M}{m}e + O(e^2)\right) \\ &= \frac{1}{m}\left(1 + M(1-1/m)e + O(e^2)\right) \end{aligned}$$

   Esto confirma nuestra sospecha:

   $$e_{n+1} = \frac{m-1}{m} e_n + O(e_n^2)$$

b) Por contradicción. Consideremos un cero x^* de $f(x)/f'(x)$. Podemos escribir $f(x) = (x-x^*)^m g(x)$ con $g(x^*) \neq 0$ y finito. Así:

$$\begin{aligned} f'(x) &= m(x-x^*)^{m-1} g(x) + (x-x^*)^m g'(x) \\ &= (x-x^*)^{m-1}(mg(x)(x-x^*)g'(x)) \end{aligned}$$

Supongamos que $f(x)/f'(x)$ tiene un cero múltiple en x^*, o sea hay h tal que $h(x^*)$ es finito y:

$$\frac{f(x)}{f'(x)} = (x-x^*)^2 h(x)$$

Simplificando:

$$\begin{aligned} \frac{f(x)}{f'(x)} &= \frac{(x-x^*)^m g(x)}{(x-x^*)^{m-1}(mg(x)+(x-x^*)g'(x))} \\ &= \frac{(x-x^*)g(x)}{mg(x)+(x-x^*)g'(x)} \end{aligned}$$

O sea:

$$\begin{aligned} \frac{(x-x^*)g(x)}{mg(x)+(x-x^*)g'(x)} &= (x-x^*)^2 h(x) \\ \frac{g(x)}{mg(x)+(x-x^*)g'(x)} &= (x-x^*)h(x) \end{aligned}$$

Pero para $x = x^*$, el lado izquierdo es $1/m \neq 0$, el lado derecho es cero. Contradicción.

c) La iteración obvia es usar Newton sobre la función indicada. Tenemos:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{f(x)}{f'(x)}\right) &= \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} \\ \frac{(f(x)/f'(x))}{(f(x)/f'(x))'} &= \frac{f(x)(f'(x))^2}{f'(x)((f'(x))^2 - f(x)f''(x))} \\ &= \frac{f(x)f'(x)}{(f'(x))^2 - f(x)f''(x)} \end{aligned}$$

La iteración resultante es:

$$x_{n+1} = x_n - \frac{f(x)f'(x)}{(f'(x))^2 - f(x)f''(x)}$$

d) No.

- Las funciones con ceros múltiples son raras en la práctica.
- La fórmula dada entrará en problemas de cancelación cerca de un cero múltiple (donde $f'(x^*) = 0$).
- Hay fórmulas no mucho más complejas que dan convergencia cúbica si el cero no es múltiple al incluir la segunda derivada.

Es preferible monitorizar el avance y tomar medidas en caso que se vea convergencia lenta.

Puntajes

Total	40
a)	12
b)	15
c)	5
d)	8

2. Por turno. Son temas independientes.

   a) Demostramos las tres propiedades.

   Elección voraz: Por contradicción. Supongamos que el monto w, con $b^k < w < b^{k+1}$, es el mínimo para el que el algoritmo voraz no da un óptimo. Sea $M(w) = (m_0, m_1, \ldots, m_n)$ el vector de las cantidades de cada tipo de moneda en la solución óptima y $G(w) = (g_0, g_1, \ldots, g_n)$ el vector de las cantidades de cada tipo de moneda en la solución voraz. Entonces $m_r = g_r = 0$ para $r > k$ y $g_k > m_k$ (porque w es mínimo; si no, podríamos reducir ambas representaciones eliminando potencias mayores). Pero en tal caso estamos representando $(g_k - m_k) \cdot b^k$ en monedas de tamaño menor a b^k, que necesariamente son más que las de denominación b^k usadas por el algoritmo voraz.

   Estructura inductiva: Si la máxima denominación usada es b^k, el algoritmo voraz entrega $\lfloor w/b^k \rfloor \cdot b^k$ en denominación b^k. El resto w mód b^k se entrega en monedas menores, en las que no interfiere lo anterior.

   Subestructura óptima: Combinar la elección voraz con la solución óptima para el resto (w mód b^k) nos da la solución óptima para el total (ver la demostración de Elección Voraz).

   Al cumplirse las tres propiedades, el algoritmo voraz entrega un óptimo.

   b) Basta exhibir un contraejemplo. Para 14, el algoritmo voraz da $10, 1, 1, 1, 1$ (5 monedas), óptimo es $7, 7$ (2 monedas).

Puntajes

Total	35
a) Tres propiedades	25
b) Contraejemplo	10

3. *a*) El algoritmo 1 es bastante obvio.

Algoritmo 1: Cálculo recursivo de coeficientes binomiales

```
function binomial(n, k)
    if k = 0 ∧ k = n then
        return 1
    else
        return binomial(n − 1, k) + binomial(n − 1, k − 1)
    end
```

b) Vemos que si $k = 0$ o $k = n$, se efectúa una única llamada; en caso contrario se efectúan dos llamadas recursivas. Llamemos $C_{n,k}$ al número de llamadas al calcular $\binom{n}{k}$. Resulta la recurrencia:

$$C_{n,k} = 1 + C_{n-1,k} + C_{n-1,k-1} \qquad C_{n,0} = C_{n,n} = 1$$

Es obvio que $C_{n,k} \geq \binom{n}{k}$.

c) El algoritmo memoizado 2 es bastante obvio. Lo que hace el algoritmo 2

Algoritmo 2: Cálculo memoizado de coeficientes binomiales

```
function binomial(n, k)
    if k = 0 ∧ k = n then
        return 1
    else if b_{n,k} not yet filled then
        b_{n,k} ← binomial(n − 1, k) + binomial(n − 1, k − 1)
        Mark b_{n,k} filled
    end
    return b_{n,k}
```

es llenar la parte relevante del triángulo de Pascal (¡eso es programación dinámica!). Si llena toda la fila k, debe calcular el siguiente número de elementos:

$$\begin{aligned}\sum_{0\leq r\leq n}(r+1) &= \sum_{1\leq r\leq n+1} r \\ &= \frac{(n+1)(n+2}{2} \\ &\sim \frac{n^2}{2}\end{aligned}$$

En realidad basta llenar un rectángulo de lados $k+1$ y $n-k+1$, para un total de:

$$(n-k+1)(k+1) \sim k(n-k)$$

Aproximando por variables continuas, el máximo de esta función se da para $k = n/2$, donde vale $n^2/4$ (la mitad de la estimación previa).

Puntajes

Total		25
a) Algoritmo	5	
b) Plantear recurrencia	10	
c) Reconocer área computada	10	

4. Veamos los dos puntos (son independientes):

   a) Son $N \times N$ multiplicaciones de un dígito para calcular los N productos parciales que se suman luego. Sumar los N productos parciales de hasta $2N$ dígitos son $2N \times (N-1) \sim 2N^2$ operaciones adicionales. En total asintóticamente son $3N^2$ operaciones. (en realidad, las sumas no son un cuadrado de $2N \times N$, sino solo un paralelogramo de lados paralelos N y altura N, el total es en realidad $\sim 2N^2$).

   b) Reducimos A $\lceil \log_2 A \rceil$ veces; estamos efectuando ese número de duplicaciones y rematamos con entre 0 y $\lceil \log_2 A \rceil$ sumas. En total:

   Sumas: Entre $\lceil \log_2 A \rceil$ y $2\lceil \log_2 A \rceil - 1$

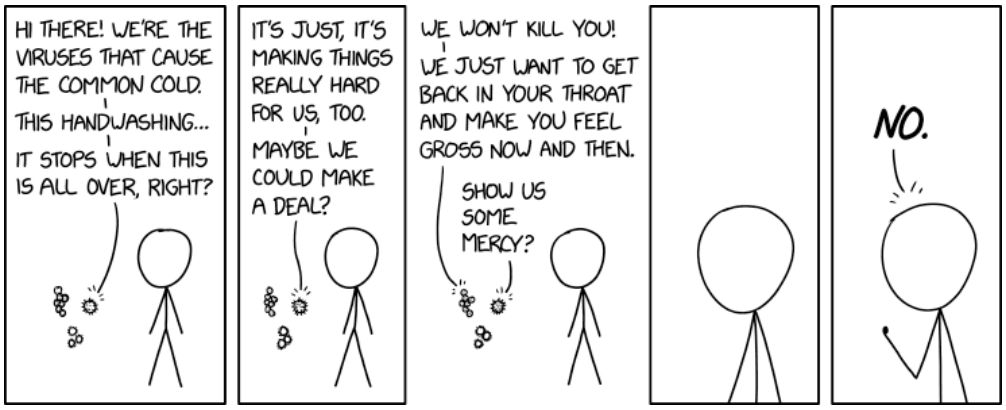
   Divisiones: Siempre $\lceil \log_2 A \rceil$

Puntajes

Total			35
a)		20	
Plantear número de multiplicaciones	10		
Estimar número de sumas	10		
b)		15	
Planteo	15		

Primer Certamen
Algoritmos y Complejidad

5 de junio de 2020



La entrega es en formato electrónico por Moodle (PDF, doc, docx). Por favor, una respuesta por página. No se aceptan páginas escritas a mano escaneadas. Nos reservamos el derecho de interrogar sobre lo entregado a un conjunto de estudiantes.

Indique en la portada de su entrega electrónica el tiempo que le tomó el desarollo (en minutos). Registre cualquier recurso externo que haya utilizado en el desarrollo de la prueba.

1. Se suele citar la ley de Moore:

   > En un plazo de 18 meses, a costo constante la capacidad del computador se duplica.

   En vista de este vertiginoso aumento de las capacidades disponibles, ¿qué sentido tiene seguir buscando algoritmos más eficientes? ¿Porqué es relevante refinar las cotas de recursos requeridos para resolver los problemas?

   (15 puntos)

2. Construya un polinomio usando su rol de a dos dígitos, agregando el dígito verificador como término constante. Si el dígito verificador es K, el término constante es 10. Por ejemplo, el rol 201.873.151-K es 2, 01, 87, 31, 15, K y da:

   $$p(x) = 2x^5 + x^4 + 87x^3 + 31x^2 + 51x + 10$$

   Siendo un polinomio de grado impar, tiene al menos un cero real. Como todos sus coeficientes son positivos, no tiene ceros positivos. Una cota simple para estos polinomios es que $-50 < x^* < 0$. Aunque probablemente es más útil una cota mayor, intentar por ejemplo con -2.

   Programe un método numérico para hallar un cero de su polinomio. Justifique el método empleado y sus valores iniciales.

   (25 puntos)

3. Explique cómo determinar los parámetros que dan el máximo grado del polinomio para el que es exacta la fórmula de cuadratura:

   $$\int_{-1}^{1} f(x)\,\mathrm{d}x = A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2) + B_0 f'(x_0) + B_1 f'(x_2)$$

   ¿Hasta qué grado de un polinomio puede ser exacta?

   (25 puntos)

4. Considere el problema de resolver sistemas lineales de n ecuaciones en n variables, que podemos anotar $\mathbf{Ax} = \mathbf{b}$, donde $\mathbf{A}$ es una matriz de $n \times n$, $\mathbf{x}$ y $\mathbf{b}$ son vectores de n elementos.

   a) Anotamos $\det(\mathbf{A})$ para el determinante de la matriz $\mathbf{A}$. Puede calcularse el determinante de $\mathbf{A}$ expandiendo por una fila (o columna). Si $\mathbf{A}_{ij}$ es la matriz $\mathbf{A}$ sin la fila i y sin la columna j, podemos escribir (expandir por la fila i):

   $$\det(\mathbf{A}) = \sum_{1 \le j \le n} (-1)^j a_{ij} \det(\mathbf{A}_{ij})$$

   El determinante de una matriz de 1×1 es simplemente el valor de su único elemento. ¿Exactamente cuántas operaciones (sumas y multiplicaciones entre elementos) se requieren para calcular $\det(\mathbf{A})$ de $n \times n$ usando el algoritmo esbozado?

   b) La regla de Cramer dice que la solución a $\mathbf{Ax} = \mathbf{b}$ es el vector de los x_j dados por:

   $$x_j = \frac{\det(\mathbf{A}_j)}{\det(\mathbf{A})}$$

   Acá $\mathbf{A}_j$ es la matriz $\mathbf{A}$ en que se substituye la columna j por el vector $\mathbf{b}$. Si el costo de calcular el determinante de una matrix de $n \times n$ es $C(n)$ (lo obtenido en la parte 4*a*), ¿cuál es el costo total de resolver el sistema?

   (35 puntos)

5. Una *rotulación graciosa* de un árbol de $n+1$ vértices es una asignación de los rótulos 0 a n a sus vértices tal que las diferencias (en valor absoluto) entre vértices vecinos sean todos los números de 1 a n. Un ejemplo es la rotulación graciosa del árbol de 8 vértices en la figura 1. Escriba un programa en el lenguaje de su elección que cuente el nú-

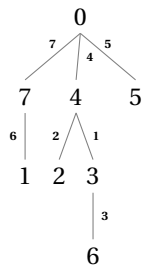


Figura 1: Una rotulación graciosa

   mero de rotulaciones graciosas del grafo P_{n+1} (un camino de $n+1$ vértices). Si d es el dígito verificador de su rol UTFSM (como en la pregunta 2), use su programa para determinar el número de rotulaciones graciosas de P_{d+7}.
   Posible pista: En Python puede usar **set** para contener rótulos o diferencias en un programa recursivo que intenta extender una solución parcial, como el programa de las ocho reinas. Por ejemplo, `A = set([k for k in range(1, 10)])` o `A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}` asignan a `A` el conjunto de los primeros diez números naturales, `for n in A` itera sobre los elementos de `A` (en un orden no especificado, pero el conjunto no debe cambiar durante la iteración), `A.add(3)` agrega 3 al conjunto `A`, `42 in A` pregunta si 42 pertenece al conjunto, `A.remove(7)` elimina 7 (si pertenece a `A`).

   (35 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

5 de junio de 2020

1. El enorme aumento de la capacidad de cómputo disponible a su vez alimenta un apetito paralelo a resolver problemas más complejos (por ejemplo, resolver modelos más realistas de situaciones físicas, procesar mayores cantidades de datos, incluso atacar problemas que antes eran imposibles de acometer). Por lo demás, hay áreas en las cuales la mejora de los algoritmos ha sido aún mayor que el aumento de capacidad de cómputo.

   Otro tema relevante es que los computadores actuales son *muy* diferentes al modelo tradicionalmente usado en el análisis: los costos de acceso a memoria no son constantes (el efecto de memorias *cache* y memoria virtual es muy relevante), hoy son comunes máquinas con múltiples procesadores, hay procesadores vectoriales de alto rendimiento –tarjetas gráficas– disponibles a costos razonables, dadas redes de alta velocidad tiene sentido distribuir el procesamiento. Estas nuevas situaciones requieren ajustes en el análisis.

   Afinar cotas teóricas siempre es relevante, indica cuándo podemos dejar de buscar algoritmos mejores al alcanzar la cota teórica.

Puntajes

Total	15
Discusión, justificación	15

2. Los polinomios son todos de la forma:

$$2x^5 + x^4 + Ax^3 + Bx^2 + Cx + D$$

Tenemos los rangos (de la –no científica– muestra de roles del semestre pasado, hay años de ingreso 2014 a 2019, códigos de carrera de 04 a 77 y suponiendo un máximo de 200 inscritos por carrera):

$$\begin{aligned} 40 \leq A \leq 97 \\ 40 \leq B \leq 72 \\ 1 \leq C \leq 99 \\ 0 \leq D \leq 10 \end{aligned}$$

Esto nos dice que para x negativo los valores están entre:

$$2x^5 + x^4 + 97x^3 + 40x^2 + 99x + 0 \leq p(x) \leq 2x^5 + x^4 + 40x^3 + 72x^2 + x + 10$$

Para $x = -1$ las cotas son $-157, 21$, para $x = -2$ son $-862, -304$, por lo que un rango que garantiza contener al menos un cero de todos los polinomios es $(-2, 0)$. El programa Raku adjunto `20201c1p3.p6` usa estos límites, pero también obtiene límites vía buscar en $-1, -2, \ldots$ el primer x que da $p(x) < 0$ y calcula el (crudo) límite dado por la página citada en el el enunciado. Provee varios métodos de búsqueda de ceros: bisección, *regula falsi*, secante, método de Newton. Ofrece calcular el polinomio para x ingresados también.

Posición personal (¡*no* es la respuesta solicitada!): el método de secante es un viejo preferido (no requiere más que el valor de la función, converge rápido si los valores iniciales son cercanos al cero); una muy buena alternativa es el método de bisección, ya que (dados puntos iniciales que acotan el cero) garantiza hallar un cero, todo esto sin requerir más que el valor de la función, y es muy sencillo de programar (aún si lento en converger).

Puntajes

Total		25
Elección del método y justificación	10	
Valor(es) inicial(es)	5	
Programa	10	
Resultado	5	

3. La fórmula contempla 8 parámetros, con lo que puede ser exacta a lo más para grado 7. Una manera de obtener los parámetros es plantear el sistema no lineal de ecuaciones:

$$\int_{-1}^{1} x^k \,\mathrm{d}x = A_0 x_0^k + A_1 x_1^k + A_2 x^k + k B_0 x_0^{k-1} + k B_2 x_2^{k-1}$$

para $0 \le k \le 7$.

Podemos simplificar el sistema notando simetrías, que hacen suponer $x_0 = -x_2$, $x_1 = 0$, $A_0 = A_2$. Simetría hace suponer que $|B_0| = |B_2|$, al menos. Con $B_0 = -B_2$ y los valores supuestos arriba las ecuaciones para k impar se reducen a $0 = 0$, vamos por buen camino. El sistema de las ecuaciones para k par es:

$$\begin{aligned}
2 &= 2A_0 + A_1 \\
\frac{2}{3} &= 2A_0 x_2^2 - 4B_0 x_2 \\
\frac{2}{5} &= 2A_0 x_2^4 - 8B_0 x_2^3 \\
\frac{2}{7} &= 2A_0 x_2^6 - 12B_0 x_2^5
\end{aligned}$$

Lamentablemente, los parámetros resultantes son complejos…

Puntajes

Total	25
Contar parámetros da el grado	10
Plantear un sistema para x^k, $0 \le k \le 7$	15
Simplificar usando simetrías	10

4. Veamos los dos puntos (son independientes):

   a) Del algoritmo esbozado resulta la siguiente recurrencia para el número de operaciones $C(n)$ en calcular un determinante de $n \times n$: como caso base, $C(1) = 0$; para calcular un determinante de $(n+1) \times (n+1)$ debemos calcular n determinantes de $n \times n$, multiplicar cada valor de éstos por un coeficiente y ejecutar $n-1$ sumas. En resumen, resulta:

$$\begin{aligned} C(1) &= 0 \\ C(n+1) &= 2n - 1 + nC(n) \end{aligned}$$

   Esta es una recurrencia lineal de primer orden, con solución simple. Un factor sumador es $1/n!$, multiplicar por el da:

$$\frac{C(n+1)}{n!} - \frac{C(n)}{(n-1)!} = \frac{2n-1}{n!}$$

   El cambio de variable $A(n) = C(n)/(n-1)!$ da:

$$\begin{aligned} A(n+1) - A(n) &= \frac{2n-1}{n!} \\ \sum_{1 \le k \le n-1} (A(k+1) - A(k)) &= \sum_{1 \le k \le n-1} \frac{2k-1}{k!} \\ A(n) - A(1) &= \\ \frac{C(n)}{(n-1)!} - \frac{C(1)}{0!} &= \\ C(n) &= (n-1)! \sum_{1 \le k \le n-1} \frac{2k-1}{k!} \end{aligned}$$

   Veamos de simplificar la suma un poco:

$$\begin{aligned} \sum_{1 \le k \le n-1} \frac{2k-1}{k!} &= 2 \sum_{1 \le k \le n-1} \frac{1}{(k-1)!} - \sum_{1 \le k \le n-1} \frac{1}{k!} \\ &= 2 \sum_{0 \le k \le n-2} \frac{1}{k!} - \sum_{1 \le k \le n-1} \frac{1}{k!} \\ &= 2 \sum_{0 \le k \le n-2} \frac{1}{k!} - \left(\sum_{0 \le k \le n-2} \frac{1}{k!} - \frac{1}{0!} + \frac{1}{(n-1)!} \right) \\ &= \sum_{0 \le k \le n-2} \frac{1}{k!} + 1 - \frac{1}{(n-1)!} \end{aligned}$$

   Podemos escribir la solución en términos de la función exponencial trunca:

$$\exp_n(x) = \sum_{0 \le k \le n} \frac{x^k}{k!}$$

quedando:

$$\begin{aligned} C(n) &= (n-1)!\left(\exp_{n-2}(1) + 1 - \frac{1}{(n-1)!}\right) \\ &\sim (\mathrm{e}+1)(n-1)! \end{aligned}$$

Maxima (ver el programa adjunto 20201c1p4a.mc) confirma este desarrollo.

b) Para resolver un sistema de $n \times n$ usando la regla de Cramer hay que calcular $n+1$ determinantes de $n \times n$ y efectuar n operaciones adicionales para calcular los x_i. El trabajo total es:

$$\begin{aligned} (n+1)C(n) + n &= (n+1)(n-1)!\left(\exp_{n-2}(1) + 1 - \frac{1}{(n-1)!}\right) \\ &\sim (\mathrm{e}+1)n! \end{aligned}$$

¡Es *mucho* trabajo!

Puntajes

Total		35
a)		20
Plantear la recurrencia para $C(n)$	10	
Resolver la recurrencia	10	
b)		15
Planteo costo total de la regla de Cramer	15	

5. Usar directamente la idea de pasar a la función recursiva los conjuntos de rótulos y diferencias sin usar lleva al programa Raku 20201c1p5.p6, que se adjunta.

   Si usamos *backtracking* (búsqueda en profundidad), una opción (similar al programa de ocho reinas en el apunte) es usar una estructura global fácil de actualizar. En este caso basta un par de arreglos que registren los rótulos y diferencias aún no empleadas. En C, requerimos las variables globales del listado 1 y tenemos la función del listado 2 que se invoca como indica el listado 3. La diferencia es que el rótulo máximo ($n-1$ en la serpiente de largo n) tiene trato especial, no lo consideramos libre ni candidato para la posición inicial.

```c
#define N 32

bool label[N];   /* Free labels */
bool diff[N];    /* Free differences */

int vertex[N];

int n;
int cnt;      /* Count of labellings found */
bool lflag;   /* List labelling? */
```

Listado 1: Variables globales

```c
void count(int k)
{
    if(k == n + 1) {
        cnt++;
        if(lflag) {
            for(int i = 0; i < n; i++)
                printf("%d, ", vertex[i]);
            printf("%d\n", vertex[n]);
        }
    }
    for(int i = 0; i <= n; i++) {
        if(label[i]) {
            int d = abs(i - vertex[k - 1]);
            if(diff[d]) {
                label[i] = diff[d] = false;
                vertex[k] = i;
                count(k + 1);
                label[i] = diff[d] = true;
            }
        }
    }
}
```

Listado 2: Función recursiva

```
cnt = 0;
for(int i = 0; i <= n; i++)
    label[i] = true;
for(int i = 1; i <= n; i++)
    diff[i] = true;
for(int i = 0; i <= n; i++) {
    label[i] = false;
    vertex[0] = i;
    count(1);
    label[i] = true;
}
printf("There are %d graceful labellings "
                    "of snake %d\n",
       cnt, n);
```

Listado 3: Programa principal (extracto)

Este programa (y suficiente paciencia) permite construir el cuadro 1.

n	Rots
0	1
1	2
2	4
3	4
4	8
5	24
6	32
7	40
8	120
9	296
10	648
11	1 328
12	3 200
13	9 912
14	25 592
15	55 920
16	143 192
17	510 696

Cuadro 1: Número de rotulaciones graciosas de P_n

Podemos usar la simetría del problema: es claro que si $v_0, v_1, \ldots, v_n$ es una rotulación graciosa de P_n, lo es $v_n, v_{n-1}, \ldots, v_0$. Por lo demás, las únicas formas

de obtener diferencia n – 0 o 0 – n, por lo que buscar otros vecinos para 0 es trabajo perdido. Uniendo ambas observaciones, podemos reducir la búsqueda forzando el orden 0 – n y manejar este caso especial. En C, no hay cambios en las variables globales del listado 1, tenemos la función del listado 4 que se invoca como indica el listado 5.

```
void count(int k)
{
    if(k == n + 1) {
        cnt += 2;
        if(lflag) {
            for(int i = 0; i < n; i++)
                printf("%d, ", vertex[i]);
            printf("%d\n", vertex[n]);
        }
    }

    if(vertex[k - 1] == 0) {
        /* Handle special case 0 -- n */
        vertex[k] = n;
        label[n] = diff[n] = false;
        count(k + 1);
        label[n] = diff[n] = true;
        }
    else {
        for(int i = 0; i < n; i++) {
            if(label[i]) {
                int d = abs(i - vertex[k - 1]);
                if(diff[d]) {
                    vertex[k] = i;
                    label[i] = diff[d] = false;
                    count(k + 1);
                    label[i] = diff[d] = true;
                }
            }
        }
    }
}
```

Listado 4: Función recursiva usando simetría

```
            cnt = 0;
            for(int i = 0; i < n; i++)
                label[i] = true;
            for(int i = 1; i < n; i++)
                diff[i] = true;
            for(int i = 0; i < n; i++) {
```

```
        vertex[0] = i;
        label[i] = false;
        count(1);
        label[i] = true;
    }
    printf("There are %d graceful labellings "
                        "of snake %d\n",
           cnt, n);
```

Listado 5: Programa principal con simetría (extracto)

Esto reduce el tiempo de ejecución de 1m1.216s a 9.645s en mi máquina, 6,3 veces en la serpiente de 14 vértices. Debiera dar ganancias mayores para largos mayores.

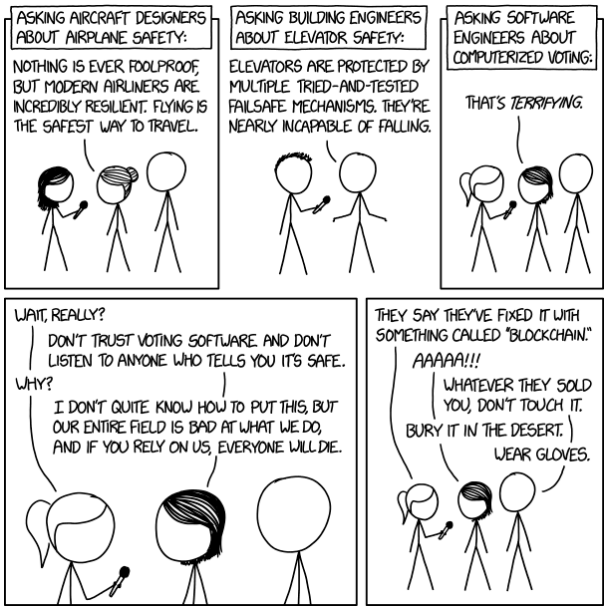
Podemos repetir el ejercicio, la diferencia $n-1$ se puede lograr solo mediante $n-1-0$ o $n-1$, y agregar eso como otro caso especial, lo que debiera dar una reducción similar, aunque en un factor mucho menor (no tiene el efecto de forzar una de cada dos rotulaciones).

Puntajes

Total	35
Planteo informal del algoritmo	5
Programa bien escrito, explicado	25
Número pedido	5

Primer Certamen
Algoritmos y Complejidad

9 de noviembre de 2019



La entrega es en formato electrónico por Moodle (PDF, doc, docx). Por favor, una respuesta por página. No se aceptan páginas escritas a mano escaneadas. Nos reservamos el derecho de interrogar sobre lo entregado a un conjunto de estudiantes.

Indique en la portada de su entrega electrónica el tiempo que le tomó el desarollo (en minutos). Registre cualquier recurso externo que haya utilizado en el desarrollo de la prueba.

1. Considere las siguientes funciones, todas las cuales para $a \geq 0$ tienen a $\sqrt{a}$ como punto fijo:

   a) $g_a(x) = (x^2 - x(1-a) - a)/(x^2 - 1)$ b) $g_b(x) = (x + a/x)/2$ c) $g_c(x) = x^2 + x - a$

   ¿Para cuáles de las anteriores converge $x_{n+1} = g(x_n)$ cerca del punto fijo? ¿Cuál converge más rápido?

   (20 puntos)

2. Explique cómo se pueden determinar los parámetros en la fórmula de cuadratura:

   $$\int_{-1}^{1} f(x)\,\mathrm{d}x = A_{-1}f(-1) + A_0 f(0) + A_1 f(1) + B_{-1}f'(-1) + B_1 f'(1)$$

   ¿Hasta qué grado de un polinomio puede ser exacta?

   (20 puntos)

3. Diseñamos un algoritmo óptimo (en términos de números de comparaciones) para hallar el máximo y el segundo mayor de los elementos de un arreglo. Extienda el algoritmo para hallar los tres mayores elementos. ¿Cuál es la complejidad?

   (25 puntos)

4. Considere el problema de las torres de Hanoi, pero siempre con dos discos del mismo tamaño. Diseñe un algoritmo para resolver este problema, moviendo $2n$ discos entre la torre de la izquierda y la de la derecha. ¿Su algoritmo deja los discos de cada par en el orden original?

   (15 puntos)

5. Se proponen las siguientes estrategias alternativas para el problema de observaciones de ALMA (cada observación i tiene instante de inicio s_i y de fin f_i, se busca una colección máxima de observaciones que no traslapan). Demuestre que son óptimas o halle un contraejemplo pequeño para cada una de ellas:

a) Elija la observación que termina última, elimine las que entran en conflicto con ésta, y repita.

b) Si no hay conflictos, programe todas las observaciones. Elimine la observación de máxima duración, y repita.

c) Si una observación x incluye completamente a una observación y, elimine x y repita. En caso contrario, elija la observación que termina última, elimine las que entran en conflicto con ésta y repita.

(30 puntos)

6. Si ponemos valor a las observaciones (la observación i tiene valor v_i e interesa maximizar la suma del valor de las observaciones programadas), diseñe un algoritmo que dé la programación óptima (si, este es uno de los ejemplos de que «algoritmos voraces no funcionan»).

(25 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

10 de enero de 2020

1. Primero analizaremos las tres funciones propuestas. Sabemos que la iteración $x_{n+1} = g(x_n)$ converge para la función diferenciable g siempre que en un intervalo $[a,b]$ que incluye el punto fijo x^* es $|g'(x)| \le c < 1$. Si $g'(x)$ es continua, esto se da en algún intervalo alrededor de x^* siempre que $|g'(x^*)| < 1$. Mientras menor sea c, más rápida la convergencia.

   En realidad, de nuestro análisis resulta:

   $$\begin{aligned} x_{n+1} - x^* &= g(x^* + (x_n - x^*)) \\ e_{n+1} &= g(x^* + e_n) - x^* \end{aligned}$$

   Si expandimos g alrededor de $x^* = g(x^*)$ obtenemos:

   $$\begin{aligned} g(x^* + e) - x^* &= \sum_{k\ge 0} \frac{1}{k!} g^{(k)}(x^*) e^k - x^* \\ &= \sum_{k\ge 1} \frac{1}{k!} g^{(k)}(x^*) e^k \end{aligned}$$

   Vale decir, si las primeras $p-1$ derivadas de g en x^* se anulan, el método es de orden p.

   Hacemos trampa: un sistema de álgebra simbólica (en este caso, nuestro regalón Maxima), expande $g(\sqrt{a}+e) - \sqrt{a}$ alrededor de $e = 0$ en cada caso, podemos leer el orden del resultado. Vea el programa adjunto `20192c1p1.mc`. De partida, nos confirma que $\sqrt{a}$ es punto fijo (el término constante de la expansión de se simplifica a 0).

   a) Para la función dada tenemos:

   $$\begin{aligned} g_a(\sqrt{a}+e) - \sqrt{a} &= -\frac{a - 2\sqrt{a} + 1}{a-1} e + O(e^2) \\ &= -\frac{(\sqrt{a}-1)^2}{(\sqrt{a}-1)(\sqrt{a}+1)} e + O(e^2) \\ &= -\frac{\sqrt{a}-1}{\sqrt{a}+1} e + O(e^2) \end{aligned}$$

   Siempre es $|\sqrt{a}-1| \le |\sqrt{a}+1|$, con igualdad solo para $a = 0$.

b) En este caso es:

$$g_b(\sqrt{a}+e)-\sqrt{a}=\frac{1}{2\sqrt{a}}e^2+O(e^3)$$

Esto tiene convergencia cuadrática.

c) Acá tenemos:

$$g_c(\sqrt{a}+e)-\sqrt{a}=(\sqrt{a}+1)e+O(e^2)$$

Diverge, ya que $.|1+\sqrt{a}|\geq 1$ para todo $a\geq 0$.

La convergencia de (b) es más que lineal, es la que converge más rápido.

Puntajes

Total		20
a)	Derivada en x^*	6
b)	Derivada en x^*	6
c)	Derivada en x^*	6
	Conclusiones	2

2. La fórmula propuesta tiene 5 parámetros, podemos plantear un sistema de 5 ecuaciones y hacerla exacta para polinomios hasta grado 4:

$$\int_{-1}^{1} x^m \,\mathrm{d}x = \left.\frac{x^{m+1}}{m+1}\right|_{-1}^{1}$$
$$\frac{2}{m+1}\,[2 \mid m]$$

Nuestro sistema de álgebra simbólica regalón (Maxima, vea `20192c1p2.mc` en los fuentes) entrega $A_{-1} = A_1 = 7/15$, $A_0 = 16/15$, $B_{-1} = -B_1 = 1/15$.

Tenemos la suerte que «accidentalmente» (en realidad, cabe esperarlo por la simetría de las ecuaciones) se cumple la equación para $m = 5$, como confirma nuestro sistema de álgebra simbólica. El método es exacto hasta polinomios de grado 5.

Puntajes

Total	20
Planteo como sistema de ecuaciones para A, B, en x_0, $x_{\pm 1}$	10
Como son 5 parámetros, grado 4	10
«Accidentalmente» grado 5 (simetría)	5

3. Es claro que el tercer elemento mayor perdió contra el mayor o contra el segundo mayor. Una manera simple es hallar el mayor de esos elementos, del análisis del apunte sabemos que el árbol binomial que construimos hay $\log_2 n$ hijos del máximo, y el segundo tiene entre 1 y $\log_2 n - 1$ hijos, lo que da un total de $2\log_2 n$ comparaciones adicionales en el peor caso.

Puntajes

Total		25
Planteo informal del algoritmo	20	
Costo adicional	5	

4. Es muy similar al algoritmo para resolver torres de Hanoi original. Para mover $2n$ discos de A a C con las restricciones dadas, debemos mover $2n-2$ discos de A a B (queremos dejar libre C para los discos mayores), mover los discos mayores de A a C, mover $2n-2$ discos de B a C.

   Si $H_2(n)$ es el número de movidas de discos con $2n$ discos, esto da la recurrencia:

   $$H_2(n) = 2H_2(n-1) + 2 \qquad H_2(0) = 0$$

   La solución de esta recurrencia (nuevamente recurrimos a Maxima, ver el programa `20192c1p4.mc`) es:

   $$H_2(n) = 2^{n+1} - 2 = 2H(n)$$

   Acá $H(n)$ es el número de movidas en la solución del problema de las torres de Hanoi.

   Pseudocódigo es el algoritmo 1. Usamos las variables *src* para el origen, *dst* para el destino, *tmp* para la aguja de paso. Se llama como hanoi2(n, A, C, B) para mover $2n$ discos (n pares de discos del mismo tamaño).

Algoritmo 1: Solución recursiva a las torres de Hanoi modificadas

```
procedure hanoi2(n, src, dst, tmp)
    if n > 0 then
        hanoi2(n − 1, src, tmp, dst))
        Mover 2 de src a dst
        hanoi2(n − 1, tmp, dst, src)
    end
```

   Viendo el algoritmo, los dos mayores se mueven una vez, quedan en orden inverso; todos los siguientes se mueven un número par de veces (al menos una vez a un temporal antes de ir al definitivo en cada ronda; hay dos llamadas recursivas que mueven los discos menores), quedan en orden. Si se quiere retener el orden, habría que mover la torre completa de A a B (no hay forma de mover los discos mayores a B sin mover todos los demás), luego de B a C (para invertir los discos mayores dos veces). Los pasos intermedios pueden efectuarse como se indica arriba. El costo en este caso es:

   $$H_2'(n) = 2H_2(n) = 4H(n)$$

Puntajes

Total		15
Planteo informal del algoritmo	5	
Recurrencia	5	
Orden de los discos	5	

5. Vemos cada propuesta por turno. Sabemos de la discusión en clases que un algoritmo voraz que entrega una solución óptima es ejecutar las tareas en orden creciente de instante de término, siempre que no hayan conflictos. Ninguna de las propuestas se parece, por lo que lo lógico es buscar contraejemplos.

   a) Para obtener un contraejemplo, basta tener una tarea larga que termine última, que traslapa varias tareas que pueden ejecutarse en secuencia. Vea la figura 1 El algoritmo propuesto elegiría t_1, eliminando t_2 y t_3,

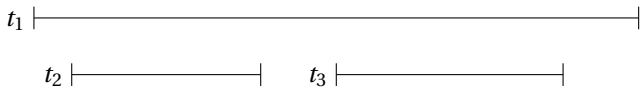


Figura 1: Contraejemplo para 5*a*

   completando 1 observación; óptimo es programar t_2 y t_3, 2 tareas.

   b) Debemos armar un contraejemplo en el que las tareas más largas sean parte de la solución óptima. Por ejemplo, vea la figura 2. El algoritmo

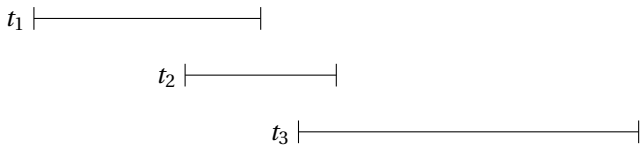


Figura 2: Contraejemplo para 5*b*

   propuesto eliminaría t_3 y t_1, dejando una única observación t_2; solución óptima es dos observaciones, t_1 y t_3.

   c) Primero, observamos que si alguna programación óptima incluye una tarea que cubre completamente a otra, podemos intercambiarlas y la programación resultante sigue siendo viable y del mismo tamaño, o sea es óptima. La primera tarea elegida por el algoritmo resulta ser la que comienza más tarde. Por simetría con el algoritmo voraz que elige las tareas en orden de término, que demostramos en clase da un óptimo, este algoritmo da un óptimo.

   Para aplicar la técnica del apunte, demostraremos las tres propiedades directamente.

   Greedy Choice: Sea Π^* una programación óptima. Sea $\hat{p}$ la observación elegida por el algoritmo propuesto. Si $\hat{p} \in \Pi^*$, estamos listos. En caso contrario, sea p la última observación programada por Π^*. Si p termina antes del comienzo de $\hat{p}$, podemos agregar $\hat{p}$ a Π^*, y Π^* no sería óptima. Por otro lado, p no puede iniciarse después del fin de $\hat{p}$, ya que sería elegida antes por el algoritmo propuesto. Vale decir, p y $\hat{p}$ traslapan. Si $\hat{p}$ está completamente contenida en p, podemos intercambiarlas dando una solución viable del mismo tamaño. Si no

está completamente contenida, no puede ser que $\hat{p}$ termine antes que p, ya que nuevamente el algoritmo propuesto hubiese considerado p antes. La única posibilidad es que p comienze antes de $\hat{p}$, como es la última tarea de Π^* nuevamente podemos intercambiarlas dando una solución viable del mismo tamaño.

Inductive substructure: Por la manera de construir el problema resultante al elegir $\hat{p}$ no hay restricciones externas: podemos combinar cualquier solución viable del problema $P - \hat{p}$ con $\hat{p}$.

Optimal substructure: Por las anteriores, es claro que se requiere una solución óptima (máximo número de observaciones programadas) para el subproblema $P - \hat{p}$ para combinar con $\hat{p}$ a una solución óptima a P.

Al cumplir las tres propiedades, este algoritmo voraz es óptimo.

Puntajes

Total		30
a) Discusión, contraejemplo	10	
b) Discusión, contraejemplo	10	
c) Discusión, demostración	10	

6. Ordenemos los instantes relevantes de tiempo (inicios s_k y fines f_k de las observaciones, y su aporte es v_k) en orden creciente, dando el arreglo t_i. Llenamos sistemáticamente el arreglo m_i con el máximo que se puede lograr hasta el instante número i (considerando que se programa alguna observación que termina en t_i o ninguna; una observación programada se combina con el óptimo que se puede lograr hasta su inicio).

   Al comienzo, no se puede haber programado nada:

   $$m_0 = 0$$

   En el instante número i, debemos obtener el conjunto de tareas que se completan en t_i, y tomar el máximo de:

   - No programar ninguna tarea, lo que se obtiene al instante anterior, m_{i-1} (no se programa ninguna de las tareas)
   - Programar cada una de las observaciones que terminan en t_i, para la tarea k con $t_i = f_k$ y $s_k = t_j$ el óptimo resulta de $v_k + m_j$.

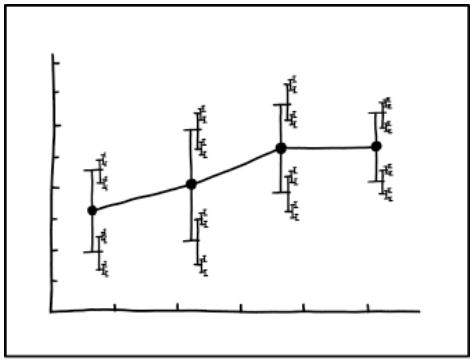
   Para poder reconstruir la programación óptima hay que registrar para cada t_i cuál de las tareas que terminan en ese instante es la que dió el óptimo.

Puntajes

Total	25
Planteo informal del algoritmo	15
Recurrencia	10

Primer Certamen
Algoritmos y Complejidad

27 de abril de 2019



1. Considere las siguientes funciones, todas las cuales tienen a $\sqrt{2}$ como punto fijo:

   a) $g_a(x) = (3x^2+2)/(4x)$ b) $g_b(x) = x/2 + 1/x$ c) $g_c(x) = x(x+1) - 2$

   ¿Para cuáles de las anteriores converge $x_{n+1} = g(x_n)$ cerca del punto fijo? ¿Cuál converge más rápido?

   (30 puntos)

2. Explique cómo se pueden determinar los parámetros A, x_{-1} y x_1 en la fórmula de cuadratura:

   $$\int_{-1}^{1} f(x)\,\mathrm{d}x = Af(x_{-1}) + Af(0) + Af(x_1)$$

   ¿Hasta qué grado de un polinomio puede ser exacta?

   (20 puntos)

3. Considere el problema de las torres de Hanoi, pero ahora con la restricción que solo pueden moverse discos entre la torre de la izquierda (A) y la del medio (B), o entre la del medio (B) y la de la derecha (C). Diseñe un algoritmo para resolver este problema, moviendo n discos entre la torre de la izquierda y la de la derecha.

   (35 puntos)

4. Se da un DAG (grafo dirigido acíclico) $G = (V, E)$, un vértice origen s y un vértice destino t. Dé un algoritmo que entregue el número de caminos de s a t en G.

   (35 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

27 de abril de 2019

1. Primero analizaremos las tres funciones propuestas. Sabemos que la iteración $x_{n+1} = g(x_n)$ converge para la función diferenciable g siempre que en un intervalo $[a,b]$ que incluye el punto fijo x^* es $|g'(x)| \leq c < 1$. Si $g'(x)$ es continua, esto se da siempre que $|g'(x^*)| < 1$. Mientras menor sea c, más rápida la convergencia.

   En realidad, de nuestro análisis resulta:

   $$\begin{aligned} x_{n+1} - x^* &= g(x^* + (x_n - x^*)) \\ e_{n+1} &= g(x^* + e_n) - x^* \end{aligned}$$

   Si expandimos g alrededor de $x^* = g(x^*)$ obtenemos:

   $$\begin{aligned} g(x^* + e) - x^* &= \sum_{k \geq 0} \frac{1}{k!} g^{(k)}(x^*) e^k - x^* \\ &= \sum_{k \geq 1} \frac{1}{k!} g^{(k)}(x^*) e^k \end{aligned}$$

   Vale decir, si las primeras $p-1$ derivadas de g en x^* se anulan, el método es de orden p.

   a) Para la función dada tenemos:

   $$\begin{aligned} g'_a(x) &= \frac{3x^2 - 2}{4x^2} \\ g'_a(x^*) &= \frac{1}{2} \end{aligned}$$

   Converge linealmente.

   b) En este caso es:

   $$\begin{aligned} g'_b(x) &= \frac{1}{2} - \frac{1}{x^2} \\ g'_b(x^*) &= 0 \end{aligned}$$

   Esto tiene convergencia más que lineal.

c) Acá tenemos:

$$\begin{aligned} g_c'(x) &= 2x + 1 \\ g_c'(x^*) &= 2\sqrt{2} + 1 \\ &\approx 3{,}828 \end{aligned}$$

Diverge.

La convergencia de (b) es más que lineal, es la que converge más rápido.

En realidad tenemos:

$$\begin{aligned} g_b''(x) &= \frac{2}{x^3} \\ g_b''(x^*) &= \frac{1}{\sqrt{2}} \\ &\neq 0 \end{aligned}$$

La convergencia de g_b es cuadrática.

Puntajes

Total		30
a)	Derivada en x^*	8
b)	Derivada en x^*	8
c)	Derivada en x^*	8
	Conclusiones	6

2. La fórmula propuesta tiene 3 parámetros, podemos plantear un sistema de 3 ecuaciones y hacerla exacta para polinomios hasta grado 2:

$$\int_{-1}^{1} x^m \,\mathrm{d}x = \left.\frac{x^{m+1}}{m+1}\right|_{-1}^{1}$$
$$\frac{2}{m+1}\,[2 \mid m]$$

Resulta el sistema:

$$\frac{2}{1} = 3A$$
$$0 = Ax_{-1} + Ax_1$$
$$\frac{2}{3} = Ax_{-1}^2 + Ax_1^2$$

De la primera ecuación tenemos:

$$A = \frac{2}{3}$$

De la segunda tenemos:

$$x_{-1} = -x_1$$

Las dos anteriores con la tercera dan:

$$1 = 2x_1^2$$
$$x_1 = \frac{1}{\sqrt{2}}$$

Tenemos la suerte que «accidentalmente» se cumple la equación para $m = 3$, dado que $x_{-1} = -x_1$:

$$0 = Ax_{-1}^3 + Ax_1^3$$

El método es exacto hasta polinomios cúbicos.

Puntajes

Total		20
Planteo como sistema de ecuaciones para A, $x_{\pm 1}$	10	
Como son 3 parámetros, grado 2	10	
«Accidentamente» grado 3 (simetría)	5	

3. Para mover n discos de A a C con las restricciones dadas, debemos mover $n-1$ discos de A a C (queremos dejar libre B para el disco mayor), mover el disco mayor de A a B, mover $n-1$ discos de C a A (debemos tener libre C para el disco mayor), mover el disco mayor de B a C, y finalmente mover $n-1$ discos de A a C. En todos los casos la torre B es de paso.

   Si $H(n)$ es el número de movidas de discos con n discos, esto da la recurrencia:

   $$H(n) = 3H(n-1) + 2 \qquad H(0) = 0$$

   La solución de esta recurrencia es:

   $$H(n) = 3^n - 1$$

   Pseudocódigo es el algoritmo 1. Usamos las variables *src* para el origen, *dst* para el destino (son A o C siempre).

Algoritmo 1: Solución recursiva a las torres de Hanoi modificadas

```
procedure hanoi*(n, src, dst)
    if n > 0 then
        hanoi*(n − 1, src, dst)
        Mover de src a B
        hanoi*(n − 1, dst, src)
        Mover de B a dst
        hanoi*(n − 1, src, dst)
    end
```

Puntajes

Total	35
Planteo informal del algoritmo	20
Recurrencia	10
Condición inicial	5

4. Este es un caso clásico de aplicación de *backtracking*. Usaremos la notación estándar de grafos, en particular $N(v)$ son los vértices vecinos a v. Al ser un grafo dirigido acíclico, no es necesario cuidarse de volver a visitar un vértice ya considerado. Resulta el algoritmo 2.

Algoritmo 2: Contar número de caminos en un DAG

```
function Paths(G, s, t)
    if s = t then
        return 0
    end
    c ← 0
    for v ∈ N(s) do
        c ← c + Paths(G, v, t)
    end
    return c
```

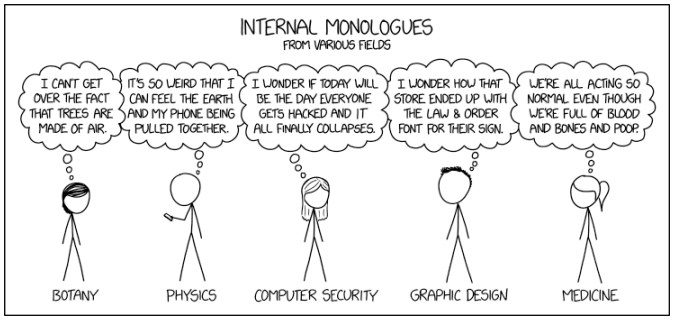
Puntajes

Total	35
Propuesta de backtracking	10
Algoritmo detallado	25

HvB/LaTeX 2ε

Primer Certamen
Algoritmos y Complejidad

27 de octubre de 2018



1. Considere la iteración de punto fijo $x_{n+1} = f(x_n)$, que converge linealmente a x^*, vale decir hay $0 < A < 1$ tal que:

$$\lim_{n\to\infty} \frac{x_{n+1} - x^*}{x_n - x^*} = A$$

   a) Suponiendo que $(x_{n+2} - x^*)/(x_{n+1} - x^*) \approx (x_{n+1} - x^*)/(x_n - x^*)$, derive una aproximación x^+ a x^*.

   b) Explique mediante un algoritmo cómo usar la aproximación de 1*a* para acelerar la convergencia.

(30 puntos)

2. El Inspector Legrasse necesita calcular numéricamente integrales del tipo siguiente en forma exacta para f cúbicas:

$$\int_0^1 x f(x)\,\mathrm{d}x$$

   Diga cómo cumplir con el mínimo trabajo (evaluaciones de la complicada función f), dando las técnicas empleadas y las fórmulas precisas (no es necesario calcular los valores).

(30 puntos)

3. Considere el problema de las torres de Hanoi, donde queremos mover n discos de la aguja A a la C, pero con la restricción que toda movida debe involucrar la aguja B (por ejemplo, no podemos simplemente mover el disco mayor de A a C, debemos pasar por B).

   a) Plantee una técnica (algoritmo) para resolver esta variante.

   b) Halle una recurrencia para el número de platos movidos por su algoritmo en términos de n.

(35 puntos)

4. Se da un DAG (grafo dirigido acíclico) $G = (V, E)$, un vértice origen s y un vértice destino t. Dé un algoritmo que entregue el número de caminos de s a t en G.

(35 puntos)

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

27 de octubre de 2018

1. Ambas partes son independientes.

   a) Bajo el supuesto indicado, podemos plantear la ecuación para la aproximación x^+:

   $$\begin{aligned}
   \frac{x_{n+2} - x^+}{x_{n+1} - x^+} &= \frac{x_{n+1} - x^+}{x_n - x^+} \\
   (x_{n+2} - x^+)(x_n - x^+) &= (x_{n+1} - x^+)^2 \\
   x_{n+2}x_n - (x_{n+2} + x_n)x^+ + (x^+)^2 &= x_{n+1}^2 - 2x_{n+1}x^+ + (x^+)^2 \\
   x_{n+2}x_n - x_{n+1}^2 &= (x_{n+2} - 2x_{n+1} + x_n)x^+
   \end{aligned}$$

   Despejando:

   $$x^+ = \frac{x_{n+2}x_n - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n}$$

   Esta fórmula es muy inestable numéricamente (el resultado es dividir dos expresiones que restan cantidades muy parecidas), puede escribirse en forma más estable como:

   $$x^+ = x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n}$$

   donde:

   $$\begin{aligned}
   \Delta x_n &= x_{n+1} - x_n \\
   \Delta^2 x_n &= \Delta(\Delta x_n) \\
   &= \Delta(x_{n+1} - x_n) \\
   &= x_{n+2} - 2x_{n+1} + x_n
   \end{aligned}$$

   Por esta fórmula se le conoce como el *proceso* Δ^2 *de Aitken*, nombrado por Alexander Aitken quien lo introdujo en 1926.

   b) Sea cual sea el resultado de la primera parte de la pregunta, resulta el algoritmo 1 para aproximar el punto fijo de $g(x)$, con el punto de partida x_0, (reemplace la fórmula para x^+ a asignar a x_0).

Algoritmo 1: Algoritmo usando aceleración de Aitken

function Aitken(g, x_0, ϵ)
  repeat
    $x_1 \leftarrow g(x_0)$
    $x_2 \leftarrow g(x_1)$
    $x_0 \leftarrow x_0 - (\Delta x_0)^2 / \Delta^2 x_0$
  until $|x_2 - x_0| \leq \epsilon$
  return x_0

Puntajes

Total		30
a) Derivar la aproximación x^+	20	
b) Algoritmo claro	10	

2. Buscamos una fórmula gaussiana exacta para polinomios hasta grado 3, por lo que requeriremos $n = 2$ puntos. La teoría nos indica buscar una secuencia de polinomios ortogonales con función de peso $w(x) = x$ sobre $[0, 1]$. La fórmula será del tipo:

$$\int_0^1 x f(x)\,\mathrm{d}x \approx A_0 f(x_0) + A_1 f(x_1)$$

Usamos el proceso de Gram-Schmidt para construir una base ortogonal, partiendo con los vectores $1, x, x^2$. El producto interno relevante es:

$$\langle f, g\rangle = \int_0^1 w(x) f(x) g(x)\,\mathrm{d}x$$

$$p_0(x) = 1$$
$$p_1(x) = x - \frac{\langle x, p_0(x)\rangle}{\langle p_0(x), p_0(x)\rangle} p_0(x)$$
$$p_2(x) = x^2 - \frac{\langle x^2, p_1(x)\rangle}{\langle p_1(x), p_1(x)\rangle} p_1(x) - \frac{\langle x^2, p_0(x)\rangle}{\langle p_0(x), p_0(x)\rangle} p_0(x)$$

Nos interesan los ceros de p_2, llamémosles x_0 y x_1.

Si llamamos $\ell_i(x)$ a los polinomios de Lagrange:

$$\ell_i(x) = \prod_{\substack{0\le k\le n\\ k\neq i}} \frac{x - x_k}{x_i - x_k}$$

los coeficientes están dados por:

$$A_i = \int_0^1 x \ell_i(x)\,\mathrm{d}x$$

Un amable sistema de álgebra simbólica (Maxima, ver el archivo `p2.mc` adjunto) entrega:

$$p_0(x) = 1$$
$$p_1(x) = x - \frac{2}{3}$$
$$p_2(x) = x^2 - \frac{6}{5}x + \frac{3}{10}$$

Los ceros de p_2 son:

$$x_0 = \frac{6 - \sqrt{6}}{10} \quad x_1 = \frac{6 + \sqrt{6}}{10}$$

Con esto los coeficientes buscados son:

$$A_0 = \frac{9 + \sqrt{6}}{36} \quad A_1 = \frac{9 - \sqrt{6}}{36}$$

Puntajes

Total		30
Planteo como integración gaussiana de 2 puntos	10	
Explicar cómo obtener $p_2(x)$	10	
Nodos son los ceros de p_2	4	
Fórmulas para coeficientes	6	

3. Por turno.

   a) La idea deberá ser:

   - Mover los $n-1$ platos superiores de A a C
   - Mover el plato mayor de A a B
   - Mover los $n-1$ platos superiores de C a A
   - Mover el plato mayor de B a C
   - Mover los $n-1$ platos superiores de A a C

   Escrito formalmente como algoritmo recursivo resulta similar al del apunte, vea el algoritmo 2. Se invoca Hanoi(n, A, C).

Algoritmo 2: Solución recursiva a la variante de las torres de Hanoi

```
procedure Hanoi(n, src, dst)
    if n > 0 then
        Hanoi(n − 1, src, dst)
        Mover de src a B
        Hanoi(n − 1, dst, src)
        Mover de B a dst
        Hanoi(n − 1, src, dst)
    end
```

   b) Llamemos H_n al número de movidas en esta variante de las torres de Hanoi. Tenemos la recurrencia:

$$H_{n+1} = 2 + 3H_n \quad H_0 = 0$$

   Definiendo:

$$h(z) = \sum_{n \geq 0} H_n z^n$$

   obtenemos la ecuación:

$$\frac{h(z) - H_0}{z} = 3h(z) + \frac{2}{1-z}$$

   Nuestro sistema de álgebra simbólica nos da h como fracciones parciales (ver el archivo `p3.mc` adjunto):

$$h(z) = \frac{1}{1-3z} - \frac{1}{1-z}$$

   de donde leemos:

$$H_n = 3^n - 1$$

Puntajes

Total		35
a) Estrategia para resolver el problema	20	
b) Derivar la recurrencia	15	

4. Este es un caso clásico de aplicación de *backtracking*. Usaremos la notación estándar de grafos, en particular $N(v)$ son los vértices vecinos a v. Resulta el algoritmo 3.

Algoritmo 3: Contar número de caminos en un DAG

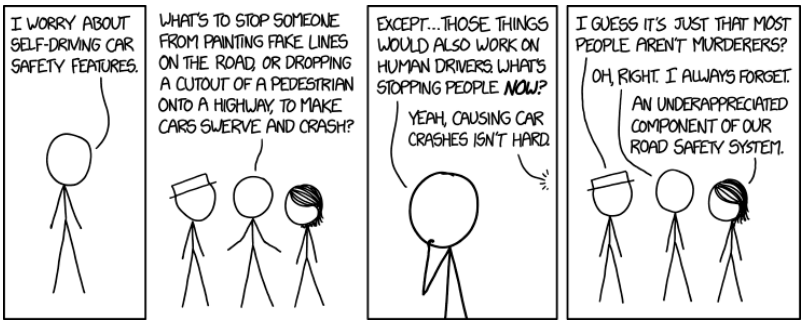
```
function Paths(G, s, t)
    if s = t then
        return 0
    end
    c ← 0
    for v ∈ N(s) do
        c ← c + Paths(G, v, t)
    end
    return c
```

Puntajes

Total	35
Propuesta de backtracking	10
Algoritmo detallado	25

Primer Certamen
Algoritmos y Complejidad

5 de mayo de 2018



1. Considere la iteración de punto fijo $x_{n+1} = 2x_n - ax_n^2$, donde a es una constante positiva. a) ¿Cuál es el punto fijo, si lo hay? b) ¿Para qué valores de a para los que hay punto fijo converge cerca del punto fijo? c) Dado un valor de a para el cual converge, ¿para qué valores iniciales x_0 converge?

   (25 puntos)

2. Siempre puede usarse la iteración de punto fijo $x_{n+1} = x_n + \alpha f(x_n)$ con x_0 adecuado para hallar un cero de f. ¿Cuál es el mejor valor posible de α, si f tiene un cero simple en x^*, y si consideramos únicamente la convergencia "cerca del cero"?

   (15 puntos)

3. Sean puntos reales $x_0 < x_1 < \ldots < x_n$. Para estimar la integral, interpolamos mediante un polinomio e integramos éste, lo que da una fórmula como:

   $$\int_{x_0}^{x_n} f(x)\,\mathrm{d}x \approx \sum_{0\le k\le n} A_k f(x_k)$$

   Con los polinomios:

   $$\ell_k(x) = \prod_{\substack{0\le j\le n \\ j\neq k}} \frac{x - x_j}{x_k - x_j}$$

   demuestre que:

   $$A_k = \int_{x_0}^{x_n} \ell_k(x)\,\mathrm{d}x$$

   (25 puntos)

4. Hay una colección de actividades, cada una de las cuales requiere ejecutarse por una unidad de tiempo en la única máquina disponible. La actividad i trae ganancia g_i si se completa antes de su plazo fatal f_i, en caso contrario no aporta nada. Se busca la secuencia de actividades a programar de forma de obtener la máxima ganancia. Demuestre que su algoritmo da una solución óptima. (30 puntos)

5. Considere la secuencia definida por:

   $$a_0 = a_1 = 1$$
   $$a_{n+2} = (3a_{n+1} + 2a_n) \text{ mód } 42$$

   a) Demuestre que la forma recursiva obvia para calcular el valor de a_n toma tiempo exponencial.

   b) Describa un algoritmo eficiente para calcular los valores de a_n.

   (35 puntos)

HvB/LaTeX2ε

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

5 de mayo de 2018

1. Por turno.

   a) Un punto fijo es la solución de la ecuación:

   $$\begin{aligned} x &= 2x - ax^2 \\ 0 &= x - ax^2 \\ &= x(1 - ax) \end{aligned}$$

   Uno de los puntos fijos es $x = 0$, el otro es $x = 1/a$.

   b) Sabemos que la iteración $x_{n+1} = g(x_n)$ converge a x^* si $|g'(x^*)| < 1$. En nuestro caso, $g(x) = 2x - ax^2$, con $g'(x) = 2 - 2ax$. Veamos los dos puntos fijos:

   $\boldsymbol{x^* = 0}$**:** Acá $g'(x^*) = 2$, nunca converge a él.

   $\boldsymbol{x^* = 1/a}$**:** Tenemos:

   $$\begin{aligned} g'(x^*) &= 2 - 2a \cdot \frac{1}{a} \\ &= 0 \end{aligned}$$

   La convergencia es superlineal.
   Nuestra expansión estándar es:

   $$\begin{aligned} g(x_n) &= g(x^*) + g'(x^*)(x_n - x^*) + \frac{1}{2} g''(x^*)(x_n - x^*)^2 \\ &\quad + O((x_n - x^*)^3) \\ x^* + e_{n+1} &= g(x^*) + g'(x^*)e_n + \frac{1}{2} g''(x^*)e_n^2 + O(e_n^3) \\ e_{n+1} &= -ae_n^2 + O(e_n^3) \end{aligned}$$

   La convergencia es cuadrática.

   c) Para que converja desde un x_0 dado, debe ser que $|x_1 - x^*| < |x_0 - x^*|$.

Veamos:

$$\begin{aligned}
|x_0 - x^*| &= \left| x_0 - \frac{1}{a} \right| \\
&= \frac{1}{a} \cdot |ax_0 - 1| \\
|x_1 - x^*| &= \left| 2x_0 - ax_0^2 - \frac{1}{a} \right| \\
&= \frac{1}{a} \cdot |2ax_0 - a^2 x_0^2 - 1| \\
&= \frac{1}{a} \cdot |a^2 x_0^2 - 2ax_0 + 1| \\
&= \frac{1}{a} \cdot (ax_0 - 1)^2
\end{aligned}$$

Una manera de comparar es mediante la razón entre los dos:

$$\begin{aligned}
\left| \frac{x_1 - x^*}{x_0 - x^*} \right| &= \frac{(ax_0 - 1)^2}{|ax_0 - 1|} \\
&= |ax_0 - 1|
\end{aligned}$$

Si ésto es menor a 1, converge desde x_0, o sea, como a es positivo:

$$\begin{aligned}
-1 < ax_0 - 1 &< 1 \\
0 < ax_0 &< 2 \\
0 < x_0 &< \frac{2}{a}
\end{aligned}$$

Para todo x_0 en este rango converge.

Puntajes

Total	25
a) Puntos fijos	8
b) Convergencia a 0 y $1/a$	8
c) Puntos iniciales x_0	9

2. Sabemos que la convergencia de iteración de punto fijo está dada por:

$$e_{n+1} = g'(x^*)e_n + O(e_n^2)$$

Por tanto, si $g'(x^*) = 0$, tenemos convergencia superlineal. En nuestro caso, eso significa:

$$g(x) = x + \alpha f(x)$$
$$g'(x^*) = 1 + \alpha f'(x^*)$$

De lo anterior:

$$\alpha = -\frac{1}{f'(x^*)}$$

Si aplicamos el mismo desarrollo que usamos al ver la convergencia del método de Newton, vemos que la convergencia es cuadrática. Claro que rara vez tenemos idea de cuánto vale $f'(x^*)$...

Puntajes

Total		15
Convergencia de FPI	5	
Anular coeficiente de e_n	8	
Valor de α	2	

3. Estamos pidiendo que la fórmula sea exacta para polinomios de grado hasta n. En particular, es exacta para los polinomios ℓ_k, de grado n. Pero estos son tal que:

   $$\ell_k(x_j) = [j = k]$$

   Para ellos la fórmula de cuadratura propuesta se reduce a:

   $$\begin{aligned}\int_{x_0}^{x_n} \ell_k(x)\,\mathrm{d}x &= \sum_{0\le j\le n} A_j \ell_k(x_j) \\ &= A_k\end{aligned}$$

   como se pedía.

   Es claro que los polinomios son linealmente independientes, por lo que podemos representar todo Π_n como combinaciones lineales de ellos.

Puntajes

Total	25
Valores de $\ell_k(x_j)$	5
Aplicar la fórmula propuesta a los ℓ_k	7
Resultado para A_k	8
Funciona con todos los polinomios por ser los ℓ_k base de Π_n	5

4. Una idea es ordenar las tareas en orden de ganancia decreciente, y programar ejecutar la tarea i justo antes de su plazo fatal si estamos a tiempo para cumplirlo. De esta forma la tarea programada interfiere lo menos posible con las demás. Si la programación resultante tiene tiempos muertos, podemos compactar al final adelantando tareas.

   Nuestro problema general es entonces que tenemos una secuencia de ranuras de tiempo, que pueden estar libres u ocupadas. Hay una secuencia de tareas, ordenadas en orden de ganancia decreciente; si hay empate en ganancias las ordenamos por plazo fatal decreciente.

   Elejimos la primera tarea de la secuencia, y la programamos en la ranura libre más tardía antes de su plazo fatal, si la hay. En caso contrario la descartamos.

   Nuestras tres propiedades son:

   Greedy Choice: Nuestro algoritmo elige la tarea $\hat{t}$ que más ganancia da de entre las que aún pueden cumplir su plazo fatal. Demostramos por contradicción que hay una solución óptima que incluye la tarea $\hat{t}$ así elegida.

   Tomemos una solución óptima. Es claro que las ranuras antes del plazo fatal de $\hat{t}$ están todas ocupadas, ya que en caso contrario podemos programar $\hat{t}$ en una libre, contradiciendo que la solución es óptima.

   Si la solución óptima incluye a $\hat{t}$, estamos listos. Si no la incluye, podemos tomar una tarea que termina antes del plazo fatal de $\hat{t}$. Si su ganancia es menor que la ganancia de $\hat{t}$, intercambiándola con $\hat{t}$ mejoramos la ganancia total, contradicción con que la solución sea óptima. La única posibilidad es que tenga la misma ganancia de $\hat{t}$, podemos intercambiarlas obteniendo una solución óptima que incluye a $\hat{t}$.

   Inductive Substructure: Sea P el problema original, $\hat{t}$ la tarea elegida por el criterio voraz, y el problema P' lo que queda al asignar $\hat{t}$ a la última ranura libre antes de su plazo fatal. Una solución viable a P', o sea, una colección de tareas a programar entre las restantes, nunca puede entrar en conflicto con la programación de $\hat{t}$; podemos combinar una solución a P' con $\hat{t}$ para dar una solución viable a P.

   En realidad, en este caso no hay restricciones "cruzadas", esto se cumple automáticamente al eliminar tareas que ya no pueden cumplir sus plazos fatales.

   Optimal Substructure: Consideremos una solución óptima Π^* al problema P. Para simplificar notación, llamemos $|\Pi|$ a la ganancia de la solución viable Π al problema P, y similarmente $|t|$ la ganancia que reporta la tarea t.

   Por **Greedy Choice**, podemos suponer sin pérdida de generalidad que Π^* incluye la elección voraz $\hat{t}$. Sea P' el problema que queda al eliminar $\hat{t}$ y las tareas que ya no pueden completarse. Sea Π' una solución óptima para P', por **Inductive Substructure** es compatible con $\hat{t}$; la ganancia de esa solución es:

   $$|\Pi'| \leq |\Pi^*| - |\hat{t}|$$

(si fuera mayor, junto con $\hat{t}$ daría una solución mejor que la óptima). Pero la solución $\Pi^* \setminus \{\hat{t}\}$ da el valor $|\Pi^*| - |\hat{t}|$, y la combinación Π' con $\hat{t}$ es óptima.

Como se cumplen las tres propiedades, el algoritmo da una solución óptima.

Puntajes

Total		30
Propuesta de algoritmo voraz	6	
Propiedad **Greedy Choice**	8	
Propiedad **Inductive Substructure**	8	
Propiedad **Optimal Substructure**	8	

5. El algoritmo recursivo obvio es el 1. Si llamamos T_n al número de llamadas a la

Algoritmo 1: Algoritmo recursivo para calcular a_n

```
function a(n)
    if n ≤ 1 then
        return 1
    else
        return (3 a(n − 1) + 2 a(n − 2)) mód 42
    end
```

función hechas para calcular a_n (el tiempo total está acotado por un múltiplo de esto), vemos que cumple:

$$T_{n+2} = T_{n+1} + T_n \qquad T_0 = T_1 = 1$$

Pero esta es la recurrencia de Fibonacci, en realidad es:

$$T_n = F_{n+1}$$

Sabemos que los números de Fibonacci crecen exponencialmente.

Alternativamente, una recurrencia de la forma:

$$U_{n+2} = \alpha U_{n+1} + \beta U_n \qquad U_0, U_1 \text{ dados}$$

tiene una solución de la forma:

$$U_n = c_1 r_1^n + c_2 r_2^n$$

donde r_1, r_2 son las raíces de la ecuación:

$$r^2 = \alpha r + \beta$$

y c_1, c_2 dependen de los valores iniciales. Si $|r_1| \geq |r_2|$, y $|r_1| > 1$, solo por una coincidencia extraordinaria es $c_1 = 0$, y:

$$|U_n| \sim |c_1 r_1^n|$$

O sea, crecimiento exponencial. En este caso la ecuación es:

$$r^2 = r + 1$$

con ceros:

$$\begin{aligned} r_{1,2} &= \frac{1 \pm \sqrt{1^2 + 4 \cdot 1 \cdot 1}}{2} \\ &= \frac{1 \pm \sqrt{5}}{2} \end{aligned}$$

Algoritmo 2: Programación dinámica para calcular a_n

```
function a(n)
    if n ≤ 1 then
        return 1
    else
        a_0 ← 1
        a_1 ← 1
        for i ← 0 to n − 1 do
            t ← (3 · a_1 + 2 · a_0) mód 42
            a_0 ← a_1
            a_1 ← t
        end
        return a_1
    end
```

Es claro que:

$$\frac{1+\sqrt{5}}{2} > 1$$

Una forma eficiente de calcular los valores es registrar los últimos dos (no se requieren los anteriores), el algoritmo 2 claramente es lineal en n.

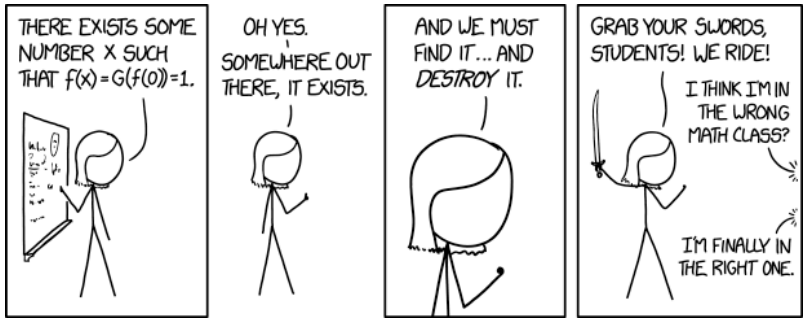
Puntajes

Total	35
Algoritmo obvio	5
Demostración que es exponencial	10
Propuesta eficiente	20

HvB/LaTeX2ε

Primer Certamen
Algoritmos y Complejidad

28 de octubre de 2017



1. Todas las funciones siguientes tienen $\sqrt{3}$ como punto fijo. ¿Cuál es la que converge más rápido cerca de $\sqrt{3}$?

$$g_a(x) = 3 - x(1-x)$$
$$g_b(x) = \frac{1}{2}x + \frac{3}{2x}$$
$$g_c(x) = \frac{x+3}{x+1}$$

(35 puntos)

2. Determine el orden de convergencia del método de Newton a x^* si la función tiene un cero doble en x^* (vale decir, $f(x^*) = f'(x^*) = 0$).

(25 puntos)

3. Suponga una fórmula de cuadratura de la forma:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = A_{-1}f(-1) + A_1 f(1) + B_{-1}f'(-1) + B_1 f'(1)$$

   a) Plantee ecuaciones para los coeficientes del polinomio interpolador de máximo grado posible en términos de $f(-1)$, $f(1)$, $f'(-1)$ y $f'(1)$.

   b) Evalúe la integral para el polinomio interpolador, y use las ecuaciones anteriores para hallar los coeficientes.

(35 puntos)

4. El problema de *ubicación de plantas* en una dimensión considera posiciones de un número finito de clientes en una recta, en las posiciones $\{c_i\}$, y un número finito de posiciones posibles de plantas, $\{p_i\}$, donde la planta i puede atender a los clientes entre las posiciones $p_i - d$ y $p_i + d$ (d es un valor fijo). Hay una potencial ubicación de planta a distancia a lo más d de cada cliente. Se busca atender a todos los clientes con el mínimo de plantas.

   a) Demuestre que la estrategia voraz de ir eligiendo ubicaciones para plantas de manera de atender al máximo número de clientes no atendidos aún no siempre da un óptimo.

   b) Demuestre que la estrategia voraz de ordenar clientes y posiciones de plantas de izquierda a derecha, y elegir siempre la posición de la planta más a la derecha capaz de atender al cliente más a la izquierda sin atención da un óptimo.

(30 puntos)

Torpedo

Proposición. *El error de interpolación mediante polinomios para los puntos distintos $x_0, \ldots, x_n \in [a, b]$ cumple:*

$$f(x) - Q_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\zeta) \prod_{0 \le j \le n} (x - x_j)$$

donde $a < \zeta < b$.

Proposición. *Si $p_n(x)$ es un polinomio mónico de grado n, entonces:*

$$\max_{-1 \le x \le 1} |p_n(x)| \ge 2^{1-n}$$

Definición. *Los polinomios de Chebyshev se definen por:*

$$T_0(x) = 1 \quad T_1(x) = x \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

Proposición. *Para $x \in [-1, 1]$:*

$$T_n(x) = \cos(n \arccos x)$$

El polinomio T_n es de grado n, y el coeficiente de x^n en T_n es 2^{n-1} si $n \ge 1$.

Definición. *Los polinomios de Lagrange para los puntos $x_0, \ldots, x_n$ son los polinomios:*

$$\ell_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} (x - x_j) / \prod_{\substack{0 \le j \le n \\ j \ne i}} (x_i - x_j)$$

Estos cumplen:

$$\ell_i(x_j) = [i = j]$$

Proposición. *El polinomio único de grado a lo más n que interpola los puntos (x_i, y_i) para $0 \le i \le n$ está dado por:*

$$Q_n(x) = \sum_{0 \le j \le n} \ell_j(x) y_j$$

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

28 de octubre de 2017

1. Sabemos que la iteración $x_{n+1} = g(x_n)$ converge a x^* si $|g'(x^*)| < 1$. Analizamos cada una de las opciones:

   a) Tenemos:

   $$\begin{aligned} g'_a(x) &= -1 + 2x \\ g'_a(\sqrt{3}) &= -1 + 2\sqrt{3} \end{aligned}$$

   Sabemos que $\sqrt{3} > 1$, con lo que $|2\sqrt{3} - 1| > 1$. Este esquema diverge.

   b) Acá es:

   $$\begin{aligned} g'_b(x) &= \frac{1}{2} - \frac{3}{2x^2} \\ g'_b(\sqrt{3}) &= \frac{1}{2} - \frac{3}{2 \cdot 3} \\ &= 0 \end{aligned}$$

   Esto indica que la convergencia es superlineal.

   En realidad, esto resulta de aplicar el método de Newton a $x^2 - 3 = 0$, la convergencia es cuadrática.

   c) Para este caso resulta:

   $$\begin{aligned} g'_c(x) &= \frac{1 \cdot (x+1) - (x+3) \cdot 1}{(x+1)^2} \\ &= \frac{-2}{(x+1)^2} \\ g'_c(\sqrt{3}) &= \frac{-2}{(1+\sqrt{3})^2} \\ &= \frac{-2}{1 + 2\sqrt{3} + 3} \\ &= \frac{-2}{4 + 2\sqrt{3}} \\ &= -\frac{1}{2 + \sqrt{3}} \end{aligned}$$

Es claro que $0 < |g_c'(\sqrt{3})| < 1$, hay convergencia lineal

Como esto es menor a 1/2, converge más rápido que bisección; pero no mucho.

Concluimos que la opción que converge más rápido es g_b.

Puntajes

Total		35
Criterio de convergencia		4
a) Derivada en $\sqrt{3}$	4	
Magnitud mayor que uno, diverge	5	
b) Derivada en $\sqrt{3}$	4	
Magnitud cero, convergencia superlineal	5	
c) Derivada en $\sqrt{3}$	4	
Magnitud menor que uno, convergencia lineal	5	
Conclusión		4

2. Expandiendo $f(x)$ y $f'(x)$ alrededor del cero x^* por Taylor:

$$\begin{aligned} f(x) &= f(x^*) + f'(x^*)(x-x^*) + \frac{1}{2}f''(x^*)(x-x^*)^2 + O((x-x^*)^3) \\ &= \frac{1}{2}f''(x^*)(x-x^*)^2 + O((x-x^*)^3) \\ f'(x) &= f''(x^*)(x-x^*) + O((x-x^*)^2) \end{aligned}$$

Llamando $e_n = x_n - x^*$, podemos escribir la iteración de Newton:

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ e_{n+1} &= e_n - \frac{\frac{1}{2}f''(x^*)e_n^2 + O(e_n^3)}{f''(x^*)e_n + O(e_n^2)} \\ &= e_n - \frac{1}{2}e_n(1 + O(e_n)) \\ &= \frac{1}{2}e_n + O(e_n^2) \end{aligned}$$

Tenemos convergencia lineal.

Puntajes

Total		25
Aproximar f, f' alrededor del cero x^*	6	
Plantear iteración de Newton en términos de e_n	7	
Simplificar expresión para el error	7	
Conclusión	5	

3. Ambas partes están relacionadas, pero las tratamos separadamente:

   a) Hay cuatro coeficientes, podemos aspirar a una fórmula exacta para polinomios hasta cúbicos:

   $$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$
   $$p'(x) = a_1 + 2a_2 x + 3a_3 x^2$$

   Si coinciden los valores de la función y las derivadas con el polinomio es:

   $$f(-1) = a_0 - a_1 + a_2 - a_3$$
   $$f(1) = a_0 + a_1 + a_2 + a_3$$
   $$f'(-1) = a_1 - 2a_2 + 3a_3$$
   $$f'(1) = a_1 + 2a_2 + 3a_3$$

   b) La integral que buscamos es:

   $$\int_{-1}^{1} p(x)\,\mathrm{d}x = \left(a_0 x + \frac{1}{2}a_1 x^2 + \frac{1}{3}a_2 x^3 + \frac{1}{4}a_3 x^4\right)\Big|_{-1}^{1}$$
   $$= 2a_0 + \frac{2}{3}a_2$$

   Sumando las primeras dos ecuaciones para los coeficientes y restando las últimas dos resulta:

   $$f(-1) + f(1) = 2a_0 + 2a_2$$
   $$f'(1) - f'(-1) = 4a_2$$

   O sea, la fórmula de cuadratura es:

   $$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx f(-1) + f(1) + \frac{1}{3}(f'(-1) - f'(1))$$

Alternativamente, vemos que la fórmula incluye 4 parármetros, con lo que a lo más esperamos que sea válida para polinomios cúbicos. Planteamos así:

$$\int_{-1}^{1} 1\,\mathrm{d}x = 2 = A_{-1}\cdot 1 + A_1 \cdot 1 + B_{-1}\cdot 0 + B_1 \cdot 0$$
$$\int_{-1}^{1} x\,\mathrm{d}x = 0 = A_{-1}\cdot (-1) + A_1 \cdot 1 + B_{-1}\cdot 1 + B_1 \cdot 1$$
$$\int_{-1}^{1} x^2\,\mathrm{d}x = \frac{2}{3} = A_{-1}\cdot 1 + A_1 \cdot 1 + B_{-1}\cdot (-2) + B_1 \cdot 2$$
$$\int_{-1}^{1} x^3\,\mathrm{d}x = 0 = A_{-1}\cdot (-1) + A_1 \cdot 1 + B_{-1}\cdot 3 + B_1 \cdot 3$$

Tenemos el sistema:

$$\begin{aligned}
A_{-1} + A_1 &= 2 \\
-A_{-1} + A_1 + B_{-1} + B_1 &= 0 \\
A_{-1} + A_1 - 2B_{-1} + 2B_1 &= 2/3 \\
-A_{-1} + A_1 + 3B_{-1} + 3B_1 &= 0
\end{aligned}$$

Restando la segunda ecuación de la tercera, vemos que $B_{-1} = -B_1$; esto en la segunda ecuación nos da $A_{-1} = A_1$, con la primera ecuación es $A_{-1} = A_1 = 1$; de la primera y la tercera ecuaciones queda $2 + 2B_1 + 2B_1 = 2/3$, o sea $B_{-1} = -B_1 = 1/3$. La cuadratura es:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = f(-1) + f(1) + \frac{1}{3}(f'(-1) - f'(1))$$

Puntajes

Total		35
Plantear el polinomio cúbico	10	
Valores, derivadas, integral del polinomio	15	
Combinar para obtener fórmula de cuadratura	10	

4. Los temas son independientes:

   a) Un ejemplo es tener plantas en $\{0,5,10\}$, con clientes en $\{1,3,4,6,7,9\}$, y $d = 4$. La estrategia voraz planteada elige la planta 5 (atiende a los cuatro clientes $3,4,6,7$, las otras atienden solo a tres), luego agrega 0 y 10 (hay que atender a 1 y 9). Estas son tres plantas; pero los clientes pueden atenderse con las dos plantas 0 y 10 únicamente.

   b) Nuestras tres propiedades son:

   Greedy Choice: En una solución óptima el cliente más a la izquierda es atendido por una planta. Si esa planta es la más a la derecha capaz de atenderlo, es nuestra elección. Si no, la elección voraz a lo menos es capaz de atender a los mismos clientes que la planta que atiende a ese cliente en la solución óptima.

   Inductive Substructure: Sea P el problema original, $\widehat{p}$ la planta elegida por el criterio voraz, y el problema P' lo que queda al eliminar $\widehat{p}$ con los clientes atendidos por $\widehat{p}$. Una solución viable a P', o sea, una colección de plantas entre las restantes, nunca puede entrar en conflicto con $\widehat{p}$, podemos combinar una solución a P' con $\widehat{p}$ para dar una solución a P.
   En realidad, en este caso simplemente no hay restricciones "cruzadas", esto se cumple automáticamente.

   Optimal Substructure: Consideremos una solución óptima Π^* al problema P. Por **Greedy Choice**, podemos suponer sin pérdida de generalidad que Π^* incluye la elección voraz $\widehat{p}$. Sea P' el problema que queda al eliminar $\widehat{p}$ y los clientes que atiende de P. Sea Π' una solución óptima para P', por **Inductive Substructure** es compatible con $\widehat{p}$; el tamaño de esa solución es:

   $$\begin{aligned} |\Pi' \cup \{\widehat{p}\}| &= |(\Pi^* \smallsetminus \{\widehat{p}\}) \cup \{\widehat{p}\}| \\ &= |\Pi^*| \end{aligned}$$

   y esta combinación es óptima.

   Como se cumplen las tres propiedades, el algoritmo da una solución óptima.

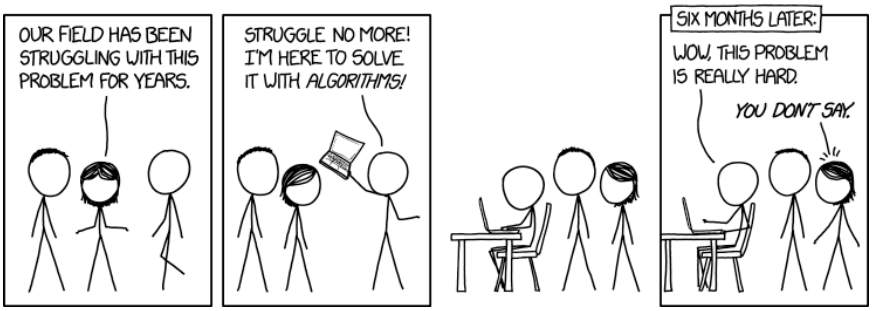
Puntajes

Total		30
a) Contraejemplo válido	10	
b) Algoritmo voraz	20	
Greedy choice	5	
Industive substructure	5	
Optimal substructure	5	
Se cumplen las propiedades, da un óptimo	5	

HvB/LaTeX 2ε

Primer Certamen
Algoritmos y Complejidad

6 de mayo de 2017



1. Siempre se puede reescribir la ecuación $f(x) = 0$ como $x = x + \alpha f(x)$ con $\alpha \neq 0$, y buscar un punto fijo x^*. ¿Cuál es el mejor valor constante de α, si x^* es un cero simple de f? ¿Cuál es el orden de convergencia con ese valor de α?

   (20 puntos)

2. Una forma alternativa de la fórmula de interpolación de Lagrange es la *baricéntrica*:

$$p_n(x) = \frac{\sum_{0 \le k \le n} \frac{w_k}{x - x_k} f(x_k)}{\sum_{0 \le k \le n} \frac{w_k}{x - x_k}}$$

   donde:

$$w_k = \left(\prod_{j \neq k} (x_k - x_j) \right)^{-1}$$

   a) Demuestre que interpola en los puntos dados (considerando los límites adecuados) amplificando y reconociendo los polinomios interpoladores de los $f(x_k)$ y de 1.

   b) Por el punto anterior, demuestre que p_n es un polinomio de grado a lo más n.

   c) ¿Cuántas operaciones en punto flotante se requieren para interpolar en un punto? Divida en precalcular valores fijos y cálculo del valor mismo.

   (25 puntos)

3. Dado un conjunto de enteros positivos $\mathscr{A} = \{a_1, a_2, \ldots, a_n\}$, determine si hay un subconjunto de $\mathscr{A}$ que suma s.
   Pista: Use programación dinámica, viendo si hay subconjuntos de $\{a_1, \ldots, a_i\}$ que suman t.

   (30 puntos)

4. Un algoritmo alternativo para construir un árbol recubridor mínimo (MST) se basa en la siguiente proposición:

   Proposición 1. *Sea C un ciclo en el grafo conexo $G = (V, E)$, con pesos de arcos $w\colon E \to \mathbb{R}^+$. Hay un árbol recubridor mínimo de G que no incluye al arco de mayor peso de C.*

   a) Demuestre la proposición 1.

   b) Use este resultado de base para un algoritmo voraz que halla un árbol recubridor mínimo.

   (25 puntos)

5. Describa brevemente y explique cuándo son aplicables las técnicas de backtracking, algoritmo voraz y programación dinámica. Contraste ventajas y desventajas de las diferentes técnicas.

   (20 puntos)

HvB/LaTeX2ε

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

6 de mayo de 2017

1. Estamos escribiendo:

   $$x = g(x) = x + \alpha f(x)$$

   Esto da la iteración:

   $$x_{n+1} = x_n + \alpha f(x_n)$$

   Expandiendo alrededor del cero x^* queda:

   $$x_{n+1} = x_n + \alpha \left(f(x^*) + f'(x^*)(x_n - x^*) + \frac{1}{2} f''(x^*)(x_n - x^*)^2 + \cdots \right)$$

   Con la substitución $e_n = x_n - x^*$, y reconociendo $f(x^*) = 0$, resulta:

   $$e_{n+1} = e_n + \alpha \left(f'(x^*) e_n + \frac{1}{2} f''(x^*) e_n^2 + \cdots \right)$$

   Para maximizar el orden de convergencia, buscamos eliminar el término en e_n, lo que se logra con $\alpha = -1/f'(x^*)$:

   $$e_{n+1} = -\frac{f''(x^*)}{2f'(x^*)} e_n^2 + \cdots$$

   lo que es convergencia cuadrática.

Puntajes

Total	20
Plantear iteración en e_n	5
Aproximar por Taylor	5
Maximizar orden de convergencia	5
Orden resultante	5

2. Cada punto por turno. Son independientes, no se requieren las soluciones de los anteriores en cada caso.

   a) La forma de Lagrange del polinomio interpolador es:

   $$L_n(x) = \sum_{0\le k\le n} \prod_{\substack{0\le j\le n \\ j\neq k}} \frac{x - x_j}{x_k - x_j} f(x_k)$$

   Definamos:

   $$w(x) = \prod_{0\le k\le n} (x - x_k)$$

   Al amplificar la expresión baricéntrica y simplificar, en el numerador queda $L_n(x)$, en el denominador queda la expresión que resulta si $f(x_k) = 1$ siempre. Como indica la pregunta, estos son los polinomios interpoladores de Lagrange de f y 1, con lo que $p_n(x_k) = f(x_k)$.

   b) Si el denominador es el polinomio interpolador de Lagrange de grado n de la función 1 en los puntos x_k, es idénticamente 1, ya que el polinomio interpolador es único.

   c) Si se requiere calcular $p_n(x)$ para varios valores de x, podemos precalcular los w_k y los $w_k f(x_k)$. Hay un total de $n(n-1)/2$ factores $x_i - x_j$ con $i \neq j$, cada w_k es el producto de $n-1$ de ellos, o sea $n-2$ multiplicaciones cada uno. Calcular $w_k f(x_k)$ es una multiplicación adicional. En resumen, la preparación calcula $2n$ valores, con un costo de $n(n-1)/2$ sumas/restas y $n(n-2) + n(n-1) = n(2n-3)$ multiplicaciones.

   Dados los valores precalculados, para evaluar $p_n(x)$ calculamos n restas $x - x_k$, $2n$ divisiones, luego $2n-2$ sumas para calcular las sumatorias, y una división final. En total, son $3n-2$ sumas/restas y $2n+1$ divisiones.

Puntajes

Total			20
a) Interpola		10	
Forma de Lagrange	4		
Amplificar por $w(x)$	3		
Reconocer los polinomios del caso	3		
b) Polinomio		5	
Polinomio interpolante es único	2		
Polinomio que interpola 1 es 1	2		
Por (c) es el polinomio pedido	1		
c) Contar operaciones		10	
Valores precalculados	5		
Cálculo del valor	5		

3. Por la pista, definimos $p[i,t]$ como verdadero si entre $\{a_1,\ldots,a_i\}$ hay un subconjunto que suma t.

   La recurrencia sobre i es bastante obvia. Para incluir a_{i+1}, podemos simplemente no usarlo (es posible obtener t con $\{a_1,\ldots,a_i\}$) o lo incluimos (debemos obtener $t-a_{i+1}$ con $\{a_1,\ldots,a_i\}$). O sea:

   $$p[i+1,t] = p[i,t] \vee p[i,t-a_{i+1}]$$

   Tenemos condiciones iniciales:

   $$p[i,t] = \begin{cases} F & \text{si } t>0 \text{ y } i=0 \\ T & \text{si } t=0 \end{cases}$$

   Es simple llevar esto al algoritmo 1 recursivo. Debemos tener cuidado de no considerar incluir a_i si $a_i > t$, como se muestra.

Algoritmo 1: Hay subconjunto de $\{a_1,\ldots,a_i\}$ con la suma t dada

```
function HaySuma(i, t)
    if t = 0 then
        return T
    else if i = 0 then
        return F
    else
        if t < a_i then
            return HaySuma(i − 1, t)
        else
            return HaySuma(i − 1, t) ∨ HaySuma(i − 1, t − a_i)
        end
    end
```

   Una solución por programación dinámica llena el arreglo desde $t=0$ hacia arriba, para cada $t=0,1,\ldots$ podemos ir llenando $p[t,i]$ sistemáticamente, con lo que los valores requeridos los habremos calculado antes. Ver el algoritmo 2. Luego de ejecutar este algoritmo, $p[n,s]$ nos dice si es posible o no la suma dada.

   Si quisiéramos además obtener un subconjunto que logra la suma pedida, habría que registrar con los $p[t,i]$ verdaderos cuál fue la opción que dio verdadero, y revisar hacia atrás desde el resultado final.

   La complejidad de este algoritmo es $O(ns\log s$, estamos calculando esencialmente ns elementos, cada uno de los cuales significa algunas operaciones entre palabras de $\log_2 s$ bits. Vimos en *Informática Teórica* (INF-155) que este problema (SUBSETSUM) es NP-completo. En términos del largo de la representación en binario este algoritmo no es polinomial, pero sí lo es en términos

Algoritmo 2: Subconjunto de $\{a_1, \ldots, a_n\}$ que suma s, programación dinámica

```
for t ← 1 to s do
    p[0, t] = F
end
for i ← 0 to n do
    p[i, 0] ← T
end
for t ← 1 to s do
    for i ← 1 to n do
        if a_i > t then
            p[i, t] ← p[i − 1, t]
        else
            p[i, t] ← p[i − 1, t] ∨ p[i − 1, t − a_i]
        end
    end
end
```

de los valores de los datos de entrada (los n elementos a_i y s). A tales algoritmos se les llama *pseudopolinomiales*.

Puntajes

Total		30
Plantear la recursión	10	
Condiciones de contorno	5	
Definir orden del cálculo de los valores	15	

4. Cada parte es independiente.

   a) La demostración es por contradicción. Consideremos un grafo $G = (V, E)$ con pesos $w\colon E \to \mathbb{R}^+$. Sea e un arco de costo máximo en un ciclo $C \leq G$, y supongamos que e necesariamente se incluye en un árbol recubridor mínimo T. Si eliminamos e de T, este se descompone en dos árboles T_1 y T_2. Como e es parte de C, hay vértices de C en T_1 y en T_2 (están los extremos de e en cada uno de ellos), por lo que hay otro arco $e' \in C$ que conecta T_1 con T_2. Vale decir, $(T \setminus \{e\}) \cup \{e'\}$ es un árbol recubridor, y $w(e') \leq w(e)$ por como se eligió e. Si $w(e') = w(e)$, podemos construir un árbol recubridor del mismo costo que no incluye e; si $w(e') \leq w(e)$ uno de menor costo. En ambos casos tenemos un árbol recubridor mínimo que no incluye e.

   b) La idea de un algoritmo voraz basado en lo anterior es considerar los arcos en orden de peso decreciente, para cada arco ver si es parte de un ciclo en el grafo actual, eliminándolo de ser así. Nuestras tres propiedades son:

   Greedy choice: Por la proposición anterior, hay un árbol recubridor mínimo que omite el arco de mayor peso que participa en un ciclo.

   Inductive substructure: Al eliminar un arco que es parte de un ciclo, el grafo resultante sigue siendo conexo, y tiene un árbol recubridor.

   Subestructura óptima: Un árbol recubridor mínimo del grafo si el arco de mayor peso que pertenece a un ciclo es un árbol recubridor mínimo del grafo completo, por la misma proposición anterior.

   Como se cumplen las tres propiedades, el algoritmo da una solución óptima.

Puntajes

Total			25
a) Demostrar el teorema		15	
b) Algoritmo		10	
Esbozar el algoritmo	4		
Demostrar las tres propiedades	6		

5. De las tres, backtracking es la más general. Siempre puede aplicarse, explorando exhaustivamente todas las posibilidades.

   Programación dinámica es aplicable siempre que el tomar una decisión restringe opciones futuras, dando lugar a problemas "más chicos" similares. A falta de un criterio que nos indique *a priori* cuál es la decisión correcta, intentamos todas las opciones y elegimos la mejor. Los problemas menores pueden resolverse recursivamente. En caso que la estructura del problema sea tal que hayan problemas menores que se repiten, podemos registrar sus soluciones para referencia futura (*memoización*) o organizar el cálculo de forma de resolver los problemas sistemáticamente en orden de tamaño creciente, de forma que cuando se requieran soluciones de problemas menores estas ya estén registradas.

   Algoritmos voraces son aplicables en situaciones como la anterior, en las cuales hay un criterio "local" (que no usa información resultante de exploración del problema) que permite decidir cuál es la elección correcta, evitando la necesidad comparar alternativas o de búsqueda.

   Comparativamente, los algoritmos voraces son simples y eficientes, pero "nunca son aplicables" (rara vez hay un criterio que permita determinar la elección correcta en forma local, y puede ocurrir que existe pero no se nos ocurre). Programación dinámica (o memoización) reduce una búsqueda extensa a algo más manejable. Programación dinámica es simple de programar, y claramente de mayor costo computacional que un algoritmo voraz. Backtracking es universalmente aplicable y simple de programar, pero el costo computacional puede ser excesivo.

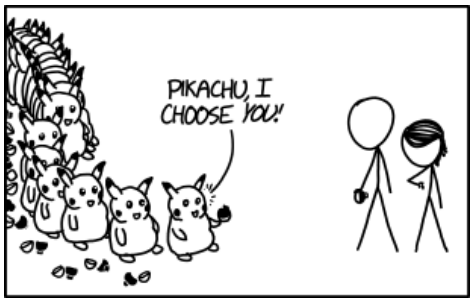
Puntajes

Total		20
Descripción de backtracking	5	
Descripción de programación dinámica	5	
Descripción de algoritmo voraz	5	
Comparar las tres	5	

HvB/LaTeX 2ε

Primer Certamen
Algoritmos y Complejidad

1 de octubre de 2016



1. Obtenga el orden de convergencia del método de Newton si x^* es un cero doble, vale decir, $f(x^*) = f'(x^*) = 0$, $f''(x^*) \neq 0$.

(20 puntos)

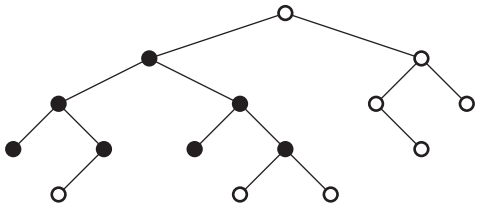


Figura 1: Un árbol binario con un máximo subárbol completo

2. Esboce un algoritmo recursivo que dado un árbol binario entregue la raíz y la altura de un máximo subárbol binario completo. En el ejemplo de la figura 1 uno de ellos está marcado en negro, su altura es 2.
Pista: Considere un recorrido en postorden.

(20 puntos)

3. La manera "obvia" de calcular el número de Fibonacci F_n es usar directamente la recurrencia:

$$F_{n+2} = F_{n+1} + F_n \qquad F_0 = 0, F_1 = 1$$

Explique cómo calcular cuántas llamadas a la función F se hacen para calcular F_n con esta estrategia.

(20 puntos)

4. Esboce un algoritmo para enumerar las permutaciones de $[1, n]$ que son desarreglos, vale decir, tales que $\pi(i) \neq i$ para todo $i \in [1, n]$.

(20 puntos)

5. Dado un arreglo a de n elementos, describa un algoritmo basado en programación dinámica para calcular la subsecuencia creciente más larga (LIS), la secuencia de índices $x_1, x_2, \ldots, x_k$ más larga tal que $a[x_1] \leq a[x_2] \leq \ldots \leq a[x_k]$. Por ejemplo, para $\langle 90, 10, 22, 9, 33, 21, 50, 41, 60, 80\rangle$ la subsecuencia creciente más larga es $\langle 10, 22, 33, 50, 60, 80\rangle$ (posiciones $2, 3, 5, 7, 9, 10$).

   a) Dé la recurrencia subyacente.
   b) Demuestre que se cumplen las suposiciones de programación dinámica.
   c) Esboce un algoritmo iterativo, indicando en particular el orden en que se llena la estructura.
   d) Indique cómo extraer la solución de su estructura.
   e) Indique la complejidad de su algoritmo.

   Pista: Considere la secuencia más larga que termina con $a[i]$.

(35 puntos)

HvB/LaTeX 2ε

Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

1 de octubre de 2016

1. La iteración del método de Newton es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Dadas nuestras condiciones sobre la función, expandiendo en serie de Taylor alrededor del cero x^* tenemos:

$$\begin{aligned} f(x) &= f(x^*) + \frac{f'(x^*)}{1!}(x-x^*) + \frac{f''(x^*)}{2!}(x-x^*)^2 + O((x-x^*)^3) \\ &= \frac{f''(x^*)}{2}(x-x^*)^2 + O((x-x^*)^3) \\ f'(x) &= f'(x^*) + \frac{f''(x^*)}{1!}(x-x^*) + O((x-x^*)^2) \end{aligned}$$

Substituyendo, llamando $e_n = x_n - x^*$ al error y absorbiendo algunas constantes en los $O(\cdot)$ queda:

$$\begin{aligned} e_{n+1} &= e_n - \frac{f''(x^*)e_n^2 + O(e_n^3)}{2f''(x^*)e_n + O(e_n^2)} \\ &= e_n\left(1 - \frac{1}{2}\right) + O(e_n^2) \end{aligned}$$

O sea, en este caso la convergencia es lineal.

Puntajes

Total	20
Iteración de Newton	5
Aproximar $f(x)$, $f'(x)$ vía Taylor	10
Orden de convergencia es lineal	5

2. La idea es que el mayor subárbol completo que tiene la raíz en un nodo dado está formado por los máximos subárboles completos con raíz en sus hijos, podados al mínimo de sus dos alturas, junto a este nodo. Mientras calculamos éstos en un recorrido en postorden, registramos en variables globales los máximos hallados hasta el momento. Resulta natural asignar "altura" -1 a los punteros nulos, da altura 0 para las hojas en forma automáticamente.

   Esto da el algoritmo siguiente:

```
maxtree ← nil
maxheight ← −1

procedure subtree (root, height)

if root = nil then
    height ← −1
else
    subtree(root− > left, hl)
    subtree(root− > right, hr)
    height ← 1 + min(hl, hr)
    if height > maxheight then
        maxheight ← height
        maxtree ← root
    end
end
```

Puntajes

Total		20
Idea general (máximo árbol en cada nodo, combinar)	5	
Caso base	2	
Registrar máximo en variables globales	3	
Algoritmo	10	

3. Lo sugerido en la pregunta corresponde a:

```
function F (n)

if n ≤ 1 then
    return 1
else
    return F (n - 1) + F (n - 2)
end
```

Llamemos t_n al número de llamadas a F al invocar $F(n)$. Se ve que si $n = 0$ o $n = 1$ se hace una llamada a F, si $n \geq 2$ se hacen $t_{n-1} + t_{n-2}$ llamadas recursivas. O sea, tenemos la recurrencia:

$$t_{n+2} = t_{n+1} + t_n \qquad t_0 = t_1 = 1$$

Para resolver esta recurrencia, recurrimos a funciones generatrices:

$$T(z) = \sum_{n \geq 0} t_n z^n$$

Aplicando las propiedades de funciones generatrices ordinarias:

$$\frac{T(z) - t_0 - t_1 z}{z^2} = \frac{T(z) - t_0}{z} + T(z) + \frac{1}{1-z}$$

Substituyendo los valores iniciales y despejando tenemos:

$$\begin{aligned} T(z) &= \frac{1 - z + z^2}{(1-z)(1-z-z^2)} \\ &= \frac{2}{1-z-z^2} - \frac{1}{1-z} \end{aligned}$$

Sabiendo que la función generatriz de los números de Fibonacci es:

$$F(z) = \frac{z}{1-z-z^2}$$

al ser $F_0 = 0$ resulta la función generatriz de los F_{n+1}:

$$\frac{F(z) - F_0}{z} = \frac{1}{1-z-z^2}$$

lo que nos permite concluir:

$$t_n = 2F_{n+1} - 1$$

Puntajes

Total		20
Plantear la función recursiva	4	
Número de llamadas en cada caso	3	
Recurrencia para t_n	6	
Condiciones iniciales	4	
Explicar cómo seguir	3	

4. Aplicando directamente la idea de backtracking resulta:

```
free ← {1, 2, ..., n}
function derangements (k)

if k = n then
    for i ← 1 to n do
        print (d [i ])
    end
else
    for i ∈ free do
        free ← free ∖ {i}
        d[k] ← i
        derangements(k + 1)
        free ← free ∪ {i}
    end
end
```

Se llama originalmente con argumento 1.

Puntajes

Total	20
Idea de backtracking	3
Caso base	5
Case general	12

5. Paso a paso.

   a) Considerando la subsecuencia creciente más larga que llega hasta $a[i]$, hay dos casos según este último:

   Es menor que todos los anteriores: En este caso, la secuencia es únicamente este elemento.

   Es mayor que algunos de los anteriores: En este caso, es la secuencia más larga que termina en un elemento anterior menor a $a[i]$ junto con $a[i]$.

   Esto da la recurrencia:

$$\text{lis}(i) = \begin{cases} 1 & \text{si } a[i] \leq a[j] \text{ para } 1 \leq j \leq i-1 \\ 1 + \max\limits_{\substack{1 \leq j \leq i-1 \\ a[j] < a[i]}} \{\text{lis}(j)\} & \end{cases}$$

   Además de el valor de $\text{lis}(i)$ debemos registrar el valor de j que da lugar al máximo para poder reconstruir la secuencia.

   b) Las suposiciones son:

   Casos exhaustivos: Consideramos todas las alternativas para el último elemento: Siempre lo incluimos, aún si es en una secuencia de largo uno.

   Subestructura inductiva: Agregar un nuevo elemento no puede interferir con las secuencias crecientes que terminan en cada uno de los elementos existentes.

   Subestructura óptima: Si la secuencia creciente más larga incluye a $a[n]$, la secuencia creciente que termina en el elemento anterior en ella claramente debe ser de largo máximo.

   c) La idea es llenar un arreglo con el largo de la secuencia más larga que llega a $a[i]$, y paralelamente registrar el valor de j que da lugar a ese máximo. Es claro que la recurrencia de 5*a* que podemos ir llenando este arreglo ordenadamente desde $i = 1$ hasta $i = n$, solo haremos uso de valores calculados anteriormente.

   Conviene ir registrando el máximo hallado hasta el momento durante el proceso de llenado, este es LIS de la secuencia.

   d) Para obtener la secuencia de lo registrado, buscamos el elemento del arreglo definido en el punto 5*c* el índice que da el máximo, y seguimos los índices j que referencian las secuencias previas.

   e) Recorrer el arreglo para hallar j según lo esbozado por la recurrencia de 5*a* tiene costo $O(i)$, como esto se repite para cada uno de los n elementos el total de esta fase es $O(n^2)$. Construir la secuencia creciente máxima de los datos registrados claramente es $O(n)$. En total, el tiempo de ejecución es $O(n^2)$.

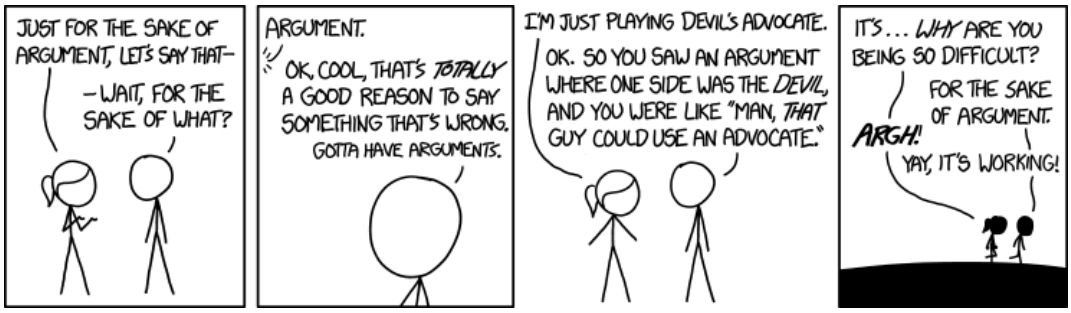
Puntajes

Total			35
a)	Recurrencia	5	
b)	Supuestos	10	
	Casos exhaustivos	3	
	Subestructura inductiva	3	
	Subestructura óptima	4	
c)	Esbozo de algoritmo	10	
d)	Extraer la subsecuencia	5	
e)	Complejidad	5	

HvB/LaTeX 2ε

Primer Certamen
Fundamentos de Informática II

14 de noviembre de 2015



1. Interesa calcular el número s_n de palabras formadas por a, b, c que no contienen símbolos repetidos (no contienen aa, bb ni cc). Explique cómo obtener una fórmula cerrada para s_n.

   (20 puntos)

2. Una permutación π tal que $\pi^2 = e$ se llama *involución*. Puede demostrarse que el número de involuciones de largo n, anotado I_n, tiene función generatriz exponencial:

   $$\begin{aligned}\widehat{I}(z) &= \sum_{n\geq 0} I_n \frac{z^n}{n!} \\ &= \mathrm{e}^{z+z^2/2}\end{aligned}$$

   Derivando $\widehat{I}(z)$ obtenga una ecuación diferencial, e interprétela en términos de una recurrencia para los I_n.

   (25 puntos)

3. Se acerca la navidad, con lo que un sansano emprendedor planea crear guirnaldas, formadas por 30 adornos navideños elegidos entre 10 posibilidades. Su investigación de mercado demuestra que se venden mejor mientras más tipos de adornos diferentes tienen. Siendo sansano, tiene curiosidad de cuántas guirnaldas formadas con el máximo número de adornos pueden crearse. Explique cómo calcular este número usando funciones generatrices.

   (20 puntos)

4. Considere el multiconjunto $\mathscr{C} = \{a_1^2, \ldots, a_n^2\}$ (n tipos de elementos, cada uno 2 veces). ¿De cuántas maneras se puede ordenar si no pueden aparecer elementos iguales adyacentes?
   Pista: Medite sobre considerar la combinación $a_i a_i$ como un nuevo símbolo A_i mientras saborea un PIE.

   (25 puntos)

5. Explique cómo calcular cuántas soluciones enteras tiene la ecuación:

   $$x_1 + x_2 + x_3 + x_4 = 12$$

   si $0 \leq x_1 \leq 3$, $0 \leq x_2 \leq 7$, $0 \leq x_3 \leq 15$, $0 \leq x_4 \leq 5$.

   (15 puntos)

6. Obtenga la función generatriz ordinaria de la suma:

   $$s_n = \sum_{0\leq k\leq n} \frac{k}{2^k}$$

   (20 puntos)

Torpedo oficial

Extracción de coeficientes

Sean $A(z) = \sum_{n\geq 0} a_n z^n$ y $B(z) = \sum_{n\geq 0} b_n z^n$:

$$[z^n]\ (\alpha A(z) + \beta B(z)) = \alpha\,[z^n]\,A(z) + \beta\,[z^n]\,B(z)$$

$$[z^n]\ A(z)\cdot B(z) = \sum_{0\leq k\leq n} a_k b_{n-k} \qquad \sum_{0\leq k\leq n} \binom{n}{k} a_k = [z^n]\ \frac{1}{1-z}\,A\Big(\frac{z}{1-z}\Big)$$

Algunas series notables

En lo que sigue, $m, n, k \in \mathbb{N}_0$, $\alpha \in \mathbb{C}$.

$$\frac{1}{1-az} = \sum_{k\geq 0} a^k z^k \qquad \frac{1-z^{m+1}}{1-z} = \sum_{0\leq k\leq m} z^k \qquad (1+z)^\alpha = \sum_{k\geq 0} \binom{\alpha}{k} z^k \qquad \frac{z}{(1-z)^2} = \sum_{n\geq 0} n z^n$$

$$\binom{\alpha}{n} = \frac{\alpha^{\underline{k}}}{k!} = \frac{\alpha(\alpha-1)\ldots(\alpha-k+1)}{k!} \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{-n}{k} = (-1)^k \binom{n+k-1}{n-1} \qquad \binom{1/2}{k} = \frac{(-1)^{k-1}}{k2^{2k-1}}\binom{2n-2}{k-1} \quad (k\geq 1) \qquad \binom{-1/2}{k} = \frac{(-1)^k}{2^{2k}}\binom{2k}{k}$$

$$e^z = \sum_{n\geq 0} \frac{z^n}{n!} \qquad e^{u+iv} = e^u(\cos v + i \sin v) \qquad \ln(1-z) = -\sum_{n\geq 1} \frac{z^n}{n}$$

Secuencias notables

Números de Fibonacci:

$$F_0 = 0 \quad F_1 = 1 \qquad F_{n+2} = F_{n+1} + F_n \qquad F_n = \frac{\tau^n - (1-\tau)^n}{\sqrt{5}} \quad \tau = \frac{1+\sqrt{5}}{2} \approx 1{,}61803$$

$$\sum_{n\geq 0} F_n z^n = \frac{z}{1-z-z^2} \qquad \sum_{n\geq 0} F_{n+1} z^n = \frac{1}{1-z-z^2}$$

Conjuntos: Cuentan número de conjuntos de k elementos elegidos entre n

$$\binom{n}{0} = 1 \quad \binom{0}{k} = [k=0] \qquad \binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k} \qquad \binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$$

$$\sum_{k,n} \binom{n}{k} x^k y^n = \frac{1}{1-(1+x)y} \qquad \sum_{n} \binom{n}{k} z^n = \frac{z^k}{(1-z)^{k+1}} \qquad \sum_{n} \binom{n+k}{k} z^n = \frac{1}{(1-z)^{k+1}}$$

Multiconjuntos: Cuentan multiconjuntos de k elementos elegidos entre n

$$\left\langle\!\binom{n}{0}\!\right\rangle = 1 \quad \left\langle\!\binom{0}{k}\!\right\rangle = [k=0] \qquad \left\langle\!\binom{n+1}{k+1}\!\right\rangle = \left\langle\!\binom{n}{k+1}\!\right\rangle + \left\langle\!\binom{n+1}{k}\!\right\rangle \qquad \left\langle\!\binom{n}{k}\!\right\rangle = \binom{n-1+k}{n-1}$$

$$\sum_{k,n} \left\langle\!\binom{n}{k}\!\right\rangle x^k y^n = \frac{1-x}{1-x-y} \qquad \sum_{k\geq 0} \left\langle\!\binom{n}{k}\!\right\rangle x^k = \frac{1}{(1-x)^n} \qquad \sum_{n\geq 0} \left\langle\!\binom{n}{k}\!\right\rangle y^n = \frac{y^{[k>0]}}{(1-y)^k}$$

Pauta de Corrección

Primer Certamen
Fundamentos de Informática II

14 de noviembre de 2015

1. Sea s_n el número de palabras de interés. Sabemos que $s_1 = 3$, y una secuencia de largo n puede extenderse de 2 formas (los símbolos que no son el último). Esto da la recurrencia:

   $$s_{n+1} = 2s_n \qquad s_1 = 3$$

   La solución a esta recurrencia es bastante obvia:

   $$s_n = s_0 \cdot 2^n$$

   Sabemos que $s_1 = 3$, con lo que:

   $$s_n = 3 \cdot 2^{n-1}$$

Puntajes

Total		20
Plantear recurrencia	10	
Solución de la recurrencia	10	

2. Tenemos que:

$$\widehat{I}(z) = \mathrm{e}^{z+z^2/2}$$
$$\widehat{I}'(z) = \mathrm{e}^{z+z^2/2} \cdot (1+z) \qquad = (1+z) \cdot \widehat{I}(z)$$

Extraemos coeficientes:

$$\begin{aligned} n![z^n]\widehat{I}'(z) &= n![z^n](1+z) \cdot \widehat{I}(z) \\ I_{n+1} &= I_n + n![z^{n-1}]\widehat{I}(z) \\ &= I_n + n! I_{n-1} \frac{1}{(n-1)!} \\ &= I_n + n I_{n-1} \end{aligned}$$

Para completar la recurrencia, necesitamos valores iniciales. Es claro que:

$$\begin{aligned} I_0 &= \frac{\widehat{I}(0)}{0!} \\ &= 1 \\ I_1 &= \frac{\widehat{I}'(0)}{1!} \\ &= \frac{(1+0)\widehat{I}(0)}{1} \\ &= 1 \end{aligned}$$

y resulta:

$$I_{n+2} = I_{n+1} + (n+1) I_n \qquad I_0 = I_1 = 1$$

Puntajes

Total	25
Derivada, en términos de $\widehat{I}(z)$	5
Extraer coeficientes	10
Recurrencia	6
Valores iniciales	4

3. Son secuencias ordenadas (los adornos están "rotulados" con su posición). Queremos que ninguno quede fuera, y para simplificar consideremos guirnaldas de cualquier largo. Cada uno de los adornos queda representado por la función generatriz:

$$\frac{z}{1!}+\frac{z^2}{2!}+\cdots=\mathrm{e}^z-1$$

Para los 10 tipos de adornos, la función generatriz es el producto, y nos interesa:

$$\begin{aligned}30![z^{30}](\mathrm{e}^z-1)^{10} &= 30![z^{30}]\sum_{0\le k\le 10}\binom{10}{k}(-1)^{10-k}\mathrm{e}^{kz}\\ &= 30!\left(\sum_{0\le k\le 10}(-1)^k\binom{10}{k}\frac{k^{30}}{30!}\right)\\ &= \sum_{0\le k\le 10}(-1)^k\binom{10}{k}k^{30}\end{aligned}$$

Da la suma:

629 137 189 252 738 366 241 450 572 800

Puntajes

Total		20
Funciones generatrices exponenciales	7	
Función generatriz para cada tipo de adorno	5	
Función generatriz de guirnaldas	3	
Extraer coeficiente de interés	5	

4. Aplicamos la receta del principio de inclusión y exclusión, usando el Tao para calcular los casos individuales:

   Universo: Todas las posibles formas de ordenar los elementos.

   Propiedades: Diremos que un ordenamiento tiene la propiedad i si los a_i están juntos.

   Resultado: Interesa el número de órdenes sin símbolos iguales juntos.

   Si hay r símbolos juntos, elegimos éstos entre n; luego por el Tao, distribuyendo $2(n-r)$ símbolos a_i y r pares:

$$\begin{aligned} N_r &= \binom{n}{r}\binom{2n-r}{\underbrace{2,2,\ldots}_{n-r},\underbrace{1,1,\ldots}_{r}} \\ &= \binom{n}{r}\frac{(2n-r)!}{(2!)^{n-r}(1!)^r} \\ &= \binom{n}{r}\frac{(2n-r)!}{2^{n-r}} \end{aligned}$$

   con lo que:

$$N(z) = \sum_{0\le r\le n}\binom{n}{r}\frac{(2n-r)!}{2^{n-r}}z^r$$

   y resulta:

$$\begin{aligned} e_0 &= N(-1) \\ &= \sum_{0\le r\le n}(-1)^r\binom{n}{r}\frac{(2n-r)!}{2^{n-r}} \\ &= (-1)^n\sum_{0\le r\le n}(-1)^r\binom{n}{r}\frac{(n+r)!}{2^r} \end{aligned}$$

   Lo último aprovechando la simetría de los coeficientes binomiales y el cambio de índice $r \mapsto n-r$.

Puntajes

Total	25
Receta del principio de inclusión y exclusión	9
Aplicar el Tao para cálculo de N_r	8
Aplicar fórmula mágica para e_0	5
Expresión	4

5. Usamos funciones generatrices. Cada variable queda representada según su rango. Como el límite para x_3 no incide, podemos omitirlo:

$$\begin{aligned}
x_1&: 1+z+z^2+z^3 && =(1-z^4)/(1-z)\\
x_2&: 1+z+z^2+z^3+z^4+z^5+z^6+z^7 && =(1-z^8)/(1-z)\\
x_3&: 1+z+z^2+\cdots && =1/(1-z)\\
x_4&: 1+z+z^2+z^3+z^4+z^5 && =(1-z^6)/(1-z)
\end{aligned}$$

Interesa:

$$\begin{aligned}
[z^{12}]\frac{(1-z^4)(1-z^6)(1-z^8)}{(1-z)^4} &= [z^{12}](1-z^4-z^6-z^8+z^{10}+z^{12}+\cdots)(1-z)^{-4}\\
&= ([z^{12}]-[z^8]-[z^6]-[z^4]+[z^2]+[z^0])(1-z)^{-4}\\
&= \binom{-4}{12}-\binom{-4}{8}-\binom{-4}{6}-\binom{-4}{4}+\binom{-4}{2}+\binom{-4}{0}\\
&= 182
\end{aligned}$$

Puntajes

Total		15
Funciones de las variables	8	
Función generatriz para número de soluciones	4	
Extraer coeficiente buscado	3	

6. Aplicar el menú de propiedades de funciones generatrices. Partimos con:

$$\sum_{n\geq 0} \frac{z^n}{2^n} = \frac{1}{1-z/2}$$
$$z\frac{\mathrm{d}}{\mathrm{d}z}\sum_{n\geq 0} \frac{z^n}{2^n} = \sum_{n\geq 0} \frac{nz^n}{2^n}$$
$$\frac{z}{2(1-z/2)} =$$
$$\frac{1}{1-z}\frac{z}{2(1-z/2)} = \sum_{n\geq 0}\left(\sum_{0\leq k\leq n} \frac{k}{2^k}\right) z^n$$

De acá:

$$\begin{aligned}\sum_{0\leq k\leq n} \frac{k}{2^k} &= [z^n]\frac{z}{(1-z)(1-z/2)} \\ &= [z^n]\left(\frac{1}{1-z} - \frac{1}{2(1-z/2)}\right) \\ &= 1 - \frac{2^{-n}}{2} \\ &= 1 - \frac{1}{2^{n+1}}\end{aligned}$$

Puntajes

Total	25
Función generatriz de 2^{-n}	7
Función generatriz de $n2^{-n}$	7
Función generatriz de $\sum n2^{-n}$	7
Extraer coeficiente para obtener la suma	4

HvB/LaTeX 2_ε